High-Level Design Tools for FPGA-Based Combinatorial Accelerators

Valery Sklyarov, Iouliia Skliarova, Pedro Almeida, Manuel Almeida

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

Abstract. Analysis of different combinatorial search algorithms has shown that they have a set of distinctive features in common. The paper suggests a number of reusable blocks that support these features and provide high-level design of combinatorial accelerators.

1 Introduction

There are many practical problems that can be formulated over such mathematical models as graphs, discrete matrices, sets, Boolean equations, etc. The majority of these models are mutually interchangeable, i.e. any of them can be selected for similar purposes. Many practical problems can be solved by applying various combinatorial search algorithms over a chosen mathematical model. Such algorithms have two distinctive features. Firstly, they require a huge number of different variants to be considered. Secondly, these variants are frequently ordered and examined with the aid of a decision tree that provides an efficient way for handling intermediate solutions. Examples of combinatorial problems that can be solved with the aid of the algorithms mentioned above are Boolean satisfiability; covering of sets and Boolean matrices; graph coloring, mapping and partitioning, etc.

This paper suggests a number of reusable blocks that can be employed for constructing application-specific hardware accelerators (co-processors) that permit combinatorial problems formulated over Boolean and ternary matrices to be solved.

2 Basic Combinatorial Search Algorithm

The considered combinatorial search algorithm is based on such primary operations that involve the application of so-called reduction and selection rules [1]. The *reduction rules* enable an initial matrix and intermediate matrices (that are constructed on each iteration through the algorithm) to be simplified. The *selection rules* allow the problem to be decomposed into several sub-problems that are examined sequentially to determine that either a solution will be found, or that no solution exists. The use of these rules for a particular example of the covering problem was considered in [2].

Search algorithms (such as [2]) for solving different combinatorial problems have similar characteristics. Their distinctive feature is the execution of problem-specific operations, traversing a decision tree starting from its root by involving such procedures as forward search and backtracking (if the forward search fails). Any branch point can be considered to be extracting a sub-tree with a local root. Thus a recursive algorithm, such as [2] can be used very efficiently. Analysis of the basic operations for general search algorithms has shown that it is reasonable to construct a set of functional blocks that enable the design process for combinatorial accelerators (co-processors) to be simplified. These blocks permit the design process to be realized at a high level of abstraction without losing sight of the details of a particular problem, and without increasing the hardware resources required or reducing performance.

The search algorithm contains branch points that can specify two or more alternative branches. The proposed computational model permits a graph to be traversed using the following strategy. All alternative branches of the search tree that might lead to the solution have to be examined. If we can prove that a selected branch does not permit a solution to be found, control is returned to the nearest branch point. All data that are needed to restore the state of any branch point on a path leading to the current step are kept in a stack. In order to store/restore data at branch points, push/pop operations are executed. Operations over vectors (rows and columns of matrices) are executed in a special functional unit that takes into account the uniqueness of combinatorial computations. Analysis of different combinatorial search algorithms has shown that the following four types of application-specific blocks are required: storage for a matrix, stack memory, a functional unit for operations over vectors, and a control circuit that supports recursion.

3 Specification of Reusable Functional Blocks

Matrix storage is the block that affects the timing characteristics of the algorithms most significantly. The proposed architecture includes memory based on RAM, mask registers that permit some rows and columns to be excluded to select a sub-matrix, and the circuits that generate RAM addresses. The latter allow the rows/columns that are not required at the current step to be skipped.

A parameterizable stack memory (that can be used for any type of intermediate data that are needed for backtracking) was constructed as a library component in VHDL and Handel-C. Elementary computations over vectors have been described using bit-manipulation operations and a reprogrammable FSM (examples can be found in [3]).

A control unit is modeled by a hierarchical FSM (HFSM) [4]. An example of a HFSM considered in [4] for executing sorting algorithms demonstrates all the steps that are required for HFSM synthesis. A number of ISE 5.2 projects for FPGAs that implement recursive hierarchical algorithms are available in the tutorial section of [5]. The same method was used for describing combinatorial search algorithms.

All the blocks considered have been designed in Handel-C and in VHDL so that their dimensions can be parameterized and they can be reused for different kinds of combinatorial accelerators (co-processors).

4 Implementation of Combinatorial Accelerators

Three combinatorial accelerators that will find a minimal row cover of a Boolean matrix using the exact algorithm [1] have been constructed based on the Handel-C and VHDL. A combinatorial accelerator that solves the Boolean satisfiability problem has been designed on the basis of VHDL. They are intended to demonstrate how the proposed library can be used. The designed macros permit storage to be allocated either in an external RAM or in FPGA embedded memory blocks (block RAM). The first accelerator was implemented on the prototyping board RC100 of Celoxica [6]. It contains 2 banks of onboard static RAM (256K*36 bit each), FPGA XC2S200 and some other components. A number of auxiliary blocks allowing data input and output have also been designed. They enable us to communicate with the keyboard and mouse, to display matrices, intermediate and final results on a VGA monitor screen, to receive/send data from/to PC, etc. The available drivers of Celoxica [6] have been utilized. However these are just 10-15% of the originally designed circuits, which in addition allow the display and scrolling of matrices, highlighting (or selecting by color) of elements considered at any intermediate step, visualizing the contents of stacks, checking any currently executing operation, and many others.

For debugging purposes the first bank of the onboard RAM was used as a memory for the VGA monitor (this bank can be used as additional storage for matrices in the "release version"). The second bank stores the matrices. The basic operation of the covering algorithm [1] is counting the number of ones in the matrix rows. This operation is only performed at the beginning of the algorithm execution. The results for all the rows are kept in FPGA block RAM and they are corrected at each newly executed step. An auxiliary stack stores these data for each branch point. General-purpose registers permit the result to be accumulated, include a counter that enables us to count the number of ones, and some temporary registers.

The Handel-C project was translated to an EDIF file in the Celoxica DK1 design suite and the bitstream for FPGA was generated from the EDIF file with the aid of Xilinx ISE 5.2.

The second accelerator was implemented on the basis of the ADM-XPL PCI board [7] containing FPGA XC2VP7 of Virtex-II Pro family. Access to the FPGA from the PC is provided through AlphaData API functions [7] that are described in the C language. Communication from the FPGA to the PC is supported by a driver described in Handel-C. Many auxiliary C++ programs that provide support for experiments and establish a user friendly graphical interface have also been implemented. The FPGA clock frequency was set to 60 MHz. An initial matrix is loaded from the PC and the accelerator solves the problem and returns the results to the PC. Finally the results are displayed on the screen. The execution time for randomly generated matrices with dimensions up to 100x100 does not exceed 2 minutes.

The third accelerator was constructed on the basis of VHDL. Two VHDL-based circuits for solving the covering and satisfiability problems were tested in Xilinx FPGAs XC4010XL and XCV812E. The reusable blocks considered above were also implemented as VHDL library modules. They have been tested in FPGA XC2S300E (the prototyping board TE-XC2Se from Trenz Electronic [8]). Many examples of such blocks are available in [5].

The results of experiments have shown that the proposed technique makes it possible to shorten the design of combinatorial accelerators (co-processors). The proposed building blocks take into account specific features of combinatorial search algorithms and they have been optimized for the problems considered. This allows block-based high-level design to be provided, i.e. to concentrate the efforts of the designer on the algorithms being considered, and avoid (or at least minimize) the details of hardware implementation. This permits consideration of either the design flow on the basis of the system-level specification language or the widely used hardware description language. The technique was validated in a number of projects that were implemented for three types of FPGAs and prototyping boards [6-8]. A set of additional blocks that were described in Handel-C and VHDL [5] provide very effective visualization and debugging tools and they can be inserted into a "debug version" and removed from the "release version". They are also very useful for experimental purposes.

5. Conclusion

It has been shown that many problems formulated over Boolean and ternary matrices can be solved with the aid of search algorithms that have a number of distinctive features. Such algorithms are recursive and they execute periodically operations for simplifying an initial matrix and making a decision for future steps. The paper suggests four primary building blocks for the high-level design of combinatorial accelerators, which provide storage and unique operations on matrices, support stacks, and implement recursive control algorithms. These blocks were described in Handel-C and in VHDL. To validate the method, three combinatorial accelerators were designed, implemented in FPGAs on the basis of stand alone and PCI boards, and tested. This work was supported by the grants FCT-PRAXIS XXI/BD/21353/99 and POSI/43140/CHS/2001.

References

1. Zakrevski, A.D.: Logical Synthesis of Cascade Networks. Moscow: Science (1981)
2. Sklyarov V., Skliarova I.: Architecture of Reconfigurable Processor for Implementing Search Algorithms over Discrete Matrices. In: Proceedings of ERSA'2003 (Engineering of Reconfigurable Systems and Algorithms) (Las Vegas, USA, 2003)
3. Sklyarov V.: Reconfigurable models of finite state machines and their implementation in FPGAs. Journal of Systems Architecture, 47 (2002) 1043-1064
4. Sklyarov, V.: Hierarchical Finite-State Machines and Their Use for Digital Control. IEEE Transactions on VLSI Systems, vol. 7, n. 2 (1999) 222-228
5. http://webct.ua.pt, "2 semester", the discipline "Computação Reconfigurável", public domain is indicated by the letter "*i*" enclosed in a circle. Login and password for access to the protected section can also be provided (via e-mails: skl@ieeta.pt, iouliia@det.ua.pt)
6. HandelC, DK1, RC100. [Online]. Available: http://www.celoxica.com
7. Alpha Data. [Online]. Available: http://www.alpha-data.com
8. Spartan-IIE Development Platform. [Online]. Available: http://www.trenz-electronic.de