A High Performance ASIC for Cellular Automata (CA) Applications

Cheryl A. Kincaid Saraju P. Mohanty Armin R. Mikler Elias Kougianos Brandon Parker
ck0017@unt.edu smohanty@cse.unt.edu mikler@cse.unt.edu eliask@unt.edu bparker@unt.edu
University of North Texas, P.O. Box 311366, Denton, TX 76203-1366, USA.

***Abstract*— CA are useful tools in modeling and simulation. However, the more complex a CA is, the longer it takes to run in typical environments. A dedicated CA machine solves this problem by allowing all cells to be computed in parallel. In this paper we present simple yet useful hardware that can perform the required computations in constant time complexity, $O(1)$.**

I. Introduction

CA are valuable tools for many scientific models and simulations [1]. They are extensively used to model population growth [2], crystalline growth [3], the spread of diseases [4], movement of particles [5], simulation of digital logic, pattern recognition, and traffic flow [6]. Various researchers have developed several custom ICs and FPGA based implementations of CA [5], [6], [7], [8].

In CA, cell state transitions are defined by a rule set and a neighborhood associated with that cell [9]. The 2 common neighborhoods in the case of a 2-D CA on a square grid are von Neumann and Moore neighborhood. Conway's *Game of Life* [8] CA is one of the best known CA rule sets. These CA are easily scalable and very useful for testing CA concepts. Hence, for this implementation we chose to use Conway's *Game of Life* to examine the benefits of a hardware versus software implementation of CA. The rules for the *Game of Life* for a 2-D array of cells using von Neumann neighborhood are: (1) Cells are either 'alive' or 'dead'. (2) If a cell has exactly 3 living neighbors, then it will also become alive during the next state transition. (3) If a cell has exactly 2 living neighbors, then its state does not change. (4) In all other cases the cell will die from 'loneliness' or 'overcrowding'.

II. Basis and Significance of Our ASIC

The computational benefit of a hardware implementation for a CA machine compared to a software implementation can be substantial. The speedup arises from two important factors. The first is an optimization of parallel processing by the distribution of calculations across the ASIC. The second speedup is derived from the inherent nature of ASIC architectures, where the circuits are designed specifically for the application without the overhead associated with general purpose CPUs. A useful metric in determining CA speed is the number of system state transitions per second [1]. While a software based machine will take multiple clock cycles per state transition, our CA machine is guaranteed to have one state transition per one clock cycle.

A comparison of the computational efficiency of our CA hardware versus a software implementation is better observed through time complexity. In a software implementation, a non-optimized CA using an $n \times m$ grid has a computational complexity of $O(n * m * k)$, where $(n * m)$ are the grid dimensions, and k is the size of the neighborhood for each cell. In our proposed hardware implementation of the cellular automaton, the computational complexity is reduced to $O(1)$, as each node calculates its next state from the given inputs of the other nodes. There is no need nor possibility to iterate over each cell in the grid, and then iterate through each cell's neighbors in order to determine the next grid state. The speed of the hardware implemented CA machine is limited only by the clock speed and settling time needed by the digital gates at each node. The clock speed of the hardware CA will in turn depend on the type of implementation, FPGA or custom IC, and corresponding technology.

III. The Proposed ASIC

We developed hardware for Conway's Game of Life CA with Von Neumann and Moore neighborhoods, but for brevity the rest of the discussion will mainly focus on the von Neumann neighborhood based implementation. Each CA machine we designed consists of 3 primary modules. The *neighborhood* module computes the number of living neighbors (0, 1, 2, 3, or more) a cell has. The *state computation* module uses this information to determine the cell's next state. The highest-level module interconnects each cell in the appropriate neighborhood configuration.

The *neighborhood* module is a combinational circuit consisting of a modified full-half adder that is designed to determine whether a cell has 0, 1, 2, 3, or more living neighbors. A logic gate level schematic for the von Neumann neighborhood is shown in Fig. 1. The current states of 3 neighbors are fed into the full adder, and the state of the 4th neighbor is used by the half adder. The results of this addition process are output as 2 bits. It may be noted that the final carry is ignored since the state "100" and "00" are both states in which the cell dies. The output bits(next state) combination are 00(Dies), 01(Dies), 10(No Change), and 11(Lives).

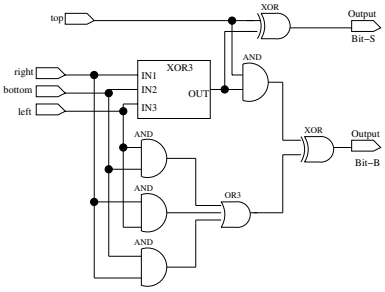

Fig. 1. The Neighborhood Module.

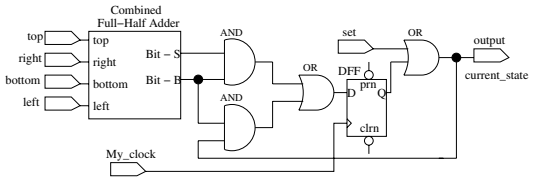

Fig. 2. The Next State Computation Module.

Fig. 3. A 4×4 Array of CA Machine using von Neumann Neighborhood

The outputs of the above combined full-half adder are used by the *state computation* module to compute a cell's next state. If the output bits are "11" or if the most significant bit is a "1" and the cell's current state is also a "1", then the cell will be alive in the next iteration. In all other cases, the cell will be dead in the next iteration. The logic gate diagram for this can be seen in Fig. 2. The result is fed into a D flip-flop where it is stored until the next clock cycle. The "set" input in Fig 2 is used as an input port to allow the initial state of the cell to be set externally. For proper operation, this input should be zeroed after the first complete clock cycle.

The highest design level in our CA machine is the lattice structure to be used. The lattice defines the actual neighborhood used by the cellular automaton. The CA machine that we present in Fig. 3 is a 16-cell, 4×4 torus array with a von Neumann neighborhood. Each cell takes as its inputs the current state of itself and each of its neighbors. Each cell uses this information to simultaneously and independently compute its next state. These results are then available for the next iteration.

IV. Results and Summary

Using Altera Quartus-II's GUI interface, 3 versions of the CA machine were created and simulated. One version uses a 4×4 von Neumann neighborhood as depicted in Fig. 3. We also created 4×4 and 7×7 Moore neighborhood versions. Simulations of the 4×4 CA machines were performed using Quartus-II with the target device being the EPF10K10LC84-3 from the FLEX10K family. Due to size, a Statix EP1S10F484C5 was the target device for the 7×7 Moore neighborhood CA. Timing simulations are shown in Fig. 4 for the 4×4 von Neumann neighborhood.

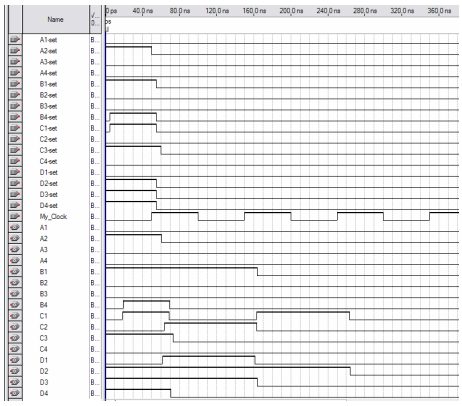

Fig. 4. Quartus II Output Waveforms for Simulation of CAM of Fig. 3

As can be seen from the timing simulation, each complete iteration of the CA completes in one clock cycle. Traditional CA are run sequentially on high-speed processors where the length of time that an iteration takes is dependent upon the size of the array of cells and the number of neighbors that each cell has. On the other hand, a dedicated CA machine such as the one discussed in this paper runs with near perfect parallelism efficiency. Thus, the number of cells makes no difference to the runtime. Each iteration runs at constant time T_{PD}, which is determined by the maximum amount of propagation delay the circuit may incur.

CA are useful tools in modeling and simulation. However, the more complex a CA is, the longer it takes to run in typical environments. A dedicated CA machine solves this problem by allowing all cells to be computed in parallel. We have presented simple yet useful hardware that can perform the computation in constant time complexity. Although the *Game of Life* has a simplistic rule-set with simplistic adjacent neighborhoods, the concepts in this paper can be expanded to other cellular automata rule-sets.

References

[1] W. A. Maniatty, et al., "Parallel Computing with Generalized Cellular Automata," `http://people.freebsd.org/~andre/`.

[2] K. C. Clarke, S. Hoppen, and L. Gaydos, "A Self-Modifying Cellular Automaton Model of Historical Urbanization in the San Francisco Bay Area," *Environment and Planning B*, vol. 24, no. 2, pp. 247–262, 1997.

[3] D. Raabe, "Cellular Automata in Materials Science with Particular Reference to Recrystallization Simulation," *Annual Review of Materials Research*, vol. 32, pp. 53–76, 2002.

[4] R. M. Z. dos Santos and S. Coutinho, "Dynamics of HIV Infection: A Cellular Automata Approach," *PRL*, vol. 87, no. 1615, October 2001.

[5] W. Heenes, et al., "FPGA Implementations of the Massively Parallel GCA Model," in *Proc of 19th IEEE IPDPS*, 2005, pp. 262b–262b.

[6] K. Nakada, T. Asai, T. Hirose, and Y. Amemiya, "Digital VLSI Implementation of Ultra-Discrete Cellular Automata for Simulating Traffic Flow," in *Proceedings of IEEE ISCIT*, 2004, pp. 394–397.

[7] T. Kobori, et al., "A Cellular Automata System with FPGA," in *Proc of 9th Annual IEEE Symposium on FCCM*, 2001, pp. 120–129.

[8] M. Waldon, "Conway's Game of Life Simulator," `http://web.mit.edu/~mwaldon/www/gameoflife.htm`.

[9] "Cellular Automata," `http://cell-auto.com/`.