Evolving FPGA Based Cellular Automata

Reid Porter and Neil Bergmann

Cooperative Research Centre for Satellite Systems,
Queensland University of Technology, Brisbane, Australia
r.porter//n.bergmann@qut.edu.edu,
Ph: +61 7 3864 1987, Fax: +61 7 3864 1361

Abstract. Cellular Automata architectures are attractive due to their fine grain parallelism, simple computational structures and local routing resources. Some researchers have used genetic algorithms to find CA that perform *useful* computations. The inherently parallel cellular automata model as well as the genetic algorithm are poorly suited to implementation on general purpose microprocessor based systems. Field Programmable Gate Arrays are an alternative that can provide significant speedup. This paper describes the Xilinx XC6216 Field Programmable Gate Array and how it is used to efficiently search a *hybrid* 2-state, 5-neighbour cellular automata rule space that exhibits computation universality. Its application to an image processing application, binary texture analysis, is discussed.

Keywords FPGA, Cellular Automata, Genetic Algorithm

1 Introduction

Cellular Automata (CA) have been considered as a model for general purpose computation by several authors. Several works describe CA capable of universal computation (computational power equivalent to a universal Turing machine [1]). Field Programmable Gate Arrays (FPGA) can provide a programmable, maximally parallel implementation of CA but can only efficiently implement small CA. Large CA require multiple FPGA and/or time multiplexing where array initialisation and result reading soon dominate the computation time. Searching CA rule spaces is one application which typically uses small test examples and therefore small CA.

Papers by Richards et. al[2], Mitchell[3] and Sahota[4] describe the use of genetic algorithms to search CA rule spaces where fitness is a function of how well the CA behaviour matches a desired algorithm output. We suggest that CA individuals be implemented in a FPGA so that the time consuming fitness evaluation task can be reduced. By using rapid reconfigurability individuals can then be swapped in and out of hardware as required by the genetic algorithm [5].

Cellular Automata have long been considered as an ideal architecture for spatially distributed/inherently parallel image processing applications [6]. We are particularly interested in the application of small CA for block based image processing applications such as local feature identification. In this paper we propose the use of XC6200 series FPGAs to accelerate exploration of CA rule spaces to find solutions to such problems.

Previous work in this area has applied genetic algorithms to fixed size CA rule tables. We consider a unique XC6200 *hybrid* CA model and search a hardware dictated rule space based on the FPGA structure. This rule space has the advantage of computation universality as well as ease of implementation.

Section 2 will introduce Field Programmable Gate Arrays and the Xilinx XC6216 device. Section 3 describes the XC6216 *hybrid* CA model and discusses the allocation of hardware resources and universal computation. Application of genetic algorithms and issues of representation are discussed in Section 4. Section 5 presents the CA Evolver experimental apparatus and its application to a block based image processing application: binary texture analysis. Results and observations will be presented in Section 6 followed by a summary and discussion of future work in Section 7.

2 Field Programmable Gate Arrays and the XC6216

Field Programmable Gate Arrays are now a popular implementation style for digital logic systems and sub-systems. These devices consist of an array of uncommitted logic gates whose function and interconnection is determined by downloading configuration information to the device. When the programming configuration is held in static RAM, the logic function implemented by those FPGAs can be dynamically reconfigured in fractions of a second by rewriting the contents of the SRAM configuration memory.

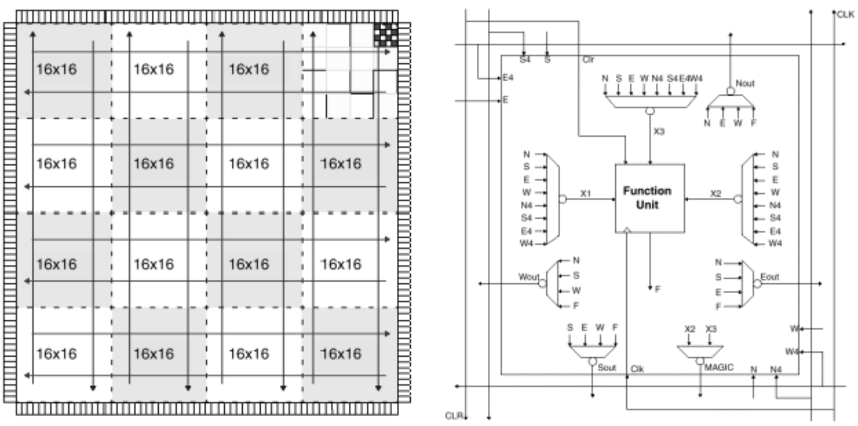


Fig. 1. (a) The XC6216 FPGA and (b) The XC6216 Cell

The XC6216 FPGA by Xilinx, depicted in Figure 1(a), has a regular 64 by 64 two-dimensional array of function blocks. These function blocks, depicted in Figure 1(b), contain a function unit capable of implementing any 2 input logic gate or 3 input multiplexer as well as a d-type flip-flop. Each block also contains local routing resources to adjacent neighbours ($N_{out}, S_{out}, E_{out}, W_{out}$ multiplexers).

Function blocks are grouped hierarchically for routing purposes. At the lowest level blocks are interconnected to the nearest neighbours in four directions. These blocks are grouped into four and supplemented with length 4 fastlanes. These 4x4 groups then form part of larger 16x16 groups which are further supplemented by length 16 wires and chip-length interconnects. The configuration memory of Xilinx XC6216 FPGA is mapped directly to the host processor enabling partial reconfigurability where part of an FPGA device can be programmed at high speed without affecting the rest of the design. Readers are referred to [7] for a more detailed description of the XC6216 device.

3 The XC6216 *hybrid* Cellular Automata

In their simplest form Cellular Automata can be considered a homogeneous array of cells in one, two, three or more dimensions. Each cell has a finite discrete state. Cells communicate with a number of local neighbours and update synchronously according to deterministic rules [8]. One of the simplest CA models has 5 neighbours and 2 states. The state of each binary 0/1 cell depends on the state of the cell at the previous time step, plus the state of the four north, south, east and west neighbours at the previous time step.

To implement this CA model, we need to be able to implement any logic function of 5 variables. This can be achieved with a look up table, or alternatively with the logic tree of Figure 2(a). This logic tree is implemented in a 4 by 4 group of XC6216 function blocks illustrated in Figure 2(b). The function block outputs, which implement the logic tree structure are illustrated by solid arrows. The CA cell state is stored in function block 5. This function block output is routed back throughout the 4 by 4 group and to all north, south, east and west faces of the cell. Its path through the local routing resources is illustrated by dashed arrows.

State information from a CA cell's north, south, east and west neighbours is routed across all function blocks through the XC6216 length 4 *fastlanes*. This means that every function block has access to all 5 boolean state variables. The 4 by 4 group of Figure 2(b) is replicated across the XC6216 to form a 15 by 15 two dimensional Cellular Automata. The North/South and East/West ends of the array are wrapped around via the XC6216 chip length routing resources.

Codd in [8] proved that a 2-state 5 neighbour CA capable of universal computation did not exist with a finite initial configuration. Several authors achieved computation-universal CA models by adding more states or larger neighbourhoods [9]. We enhance the computational power of Figure 2(a) by including the additional memory resources (flip-flops) that are freely available in each XC6216 function block. This allows us to consider a CA model based on neighbourhood information from further back in time. With this in mind, unbounded but boundable propagation is easily demonstrated. We have also demonstrated the

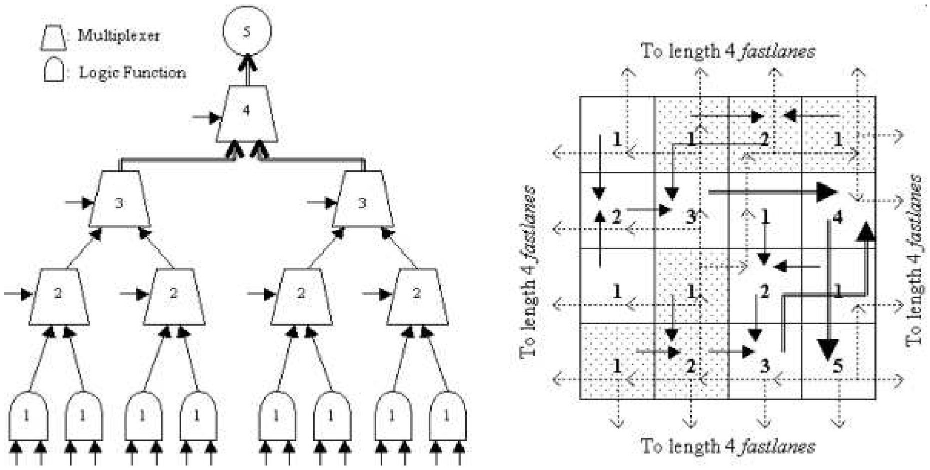


Fig. 2. (a) Flexible Logic Tree (b) Function Block Assignment

computation universality of this *hybrid* CA model by simulating Minsky's two register machine [9].

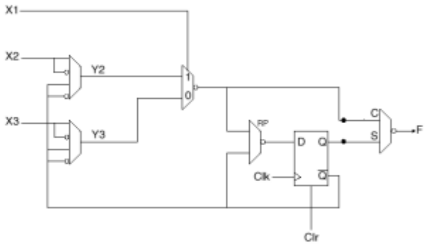
4 Searching the XC6216 *hybrid* CA rule space

To search the XC6216 *hybrid* CA rule space with a genetic algorithm several choices must be made concerning representation. In similar experiments, based in software, individuals represent fixed size rule tables. Ease of implementation leads us to consider a hardware dictated rule space with individuals based on the XC6216 configuration bit string. We consider two different hardware representations.

FPGA Bitstring: The first is based on the raw configuration bitstring of the 4 by 4 group of XC6216 function blocks. Connectivity between each group is pre-defined (to implement the 15 by 15 two dimensional CA array) but all routing and functionality within the group varies according to the genetic algorithm.

Logic Tree: The second representation is based on the flexible logic tree of Figure 2(a). In this case the routing resources both between and within each 4 by 4 group of function blocks are pre-defined and we evolve only function. A performance comparison of the *FPGA Bitstring* and *Logic Tree* hardware representations, as well as a software based CA rule table representation is presented in Section 6.

The function unit in each XC6216 function block is illustrated in Figure 3(a) and defined by two eight bit configuration bytes of Figure 3(b). CS defines whether or not the function unit will make use of the flip-flop resource. The X1, X2 and X3 configuration bits define the input signals to the logic gate or multiplexer. Y2 and Y3 define the gate or multiplexer type. RP sets the flip-flop as read-only and M defines additional routing resources.



Configuration Byte	DATA BIT							
	7	6	5	4	3	2	1	0
1	CS	X1[2:0]			X2[1:0]		X3[1:0]	
2	M	RP	Y2[1:0]		Y3[1:0]		X3[2]	X2[2]
3	North		East		West		South	

Fig. 3. (a) XC6216 Function Unit and (b) Configuration Bytes

When using the *Logic Tree* representation a CA individual is completely specified by defining these two bytes for each function block within the 4 by 4 group. The effective genetic string length for the *Logic Tree* GA is therefore 32 bytes or 256 bits. The *FPGA Bitstring* GA requires an additional byte per function block. Configuration byte 3, illustrated in Figure 3(b), defines the local $N_{out}, S_{out}, E_{out}, W_{out}$ multiplexers of Figure 1(b). The effective genetic string length for the *FPGA Bitstring* GA is therefore 48 bytes or 384 bits.

5 The CA Evolver Experimental apparatus

Our experiments are based on the HotWorks development system. This system incorporates a Xilinx XC6216 FPGA that communicates to a host processor through a 32 bit PCI bus. The main genetic algorithm loop and genetic operators are implemented on the host processor. A population of CA individuals are held in memory and downloaded into the XC6216 as required to perform the fitness evaluation task.

The reproduction scheme is similar to that used in [3]. The fittest 100 CA individuals from a population of 300 are copied directly to the next generation. The remaining 200 are generated by application of the cross-over and mutation operators on parents chosen with replacement from the *elite* 100.

For the *FPGA Bitstring* GA a one point cross-over is applied to the 384 bit configuration bit string. For the *Logic Tree* representation a specialised cross-over that constrains the structure of CA offspring to the logic tree of Figure 2(a) is used. A cross-over branch is randomly selected and then a sub-tree formed by all child nodes and branches. This sub-tree is then exchanged between two parents similar to genetic programming [5]. In both cases offspring are mutated in exactly four randomly chosen positions.

The XC6216 CA Evolver system is applied to a binary texture analysis problem which involves identifying a particular pattern within a 15 by 15 pixel area. The binary pattern chosen in this experiment is diagonal lines. Two screenshots of the CA Evolver system, each with training image on the left and ideal image on the right, are depicted in Figure 4. Each training image is *segmented* either horizontally as in Figure 4(a) or vertically 4(b).The non-patterned segment

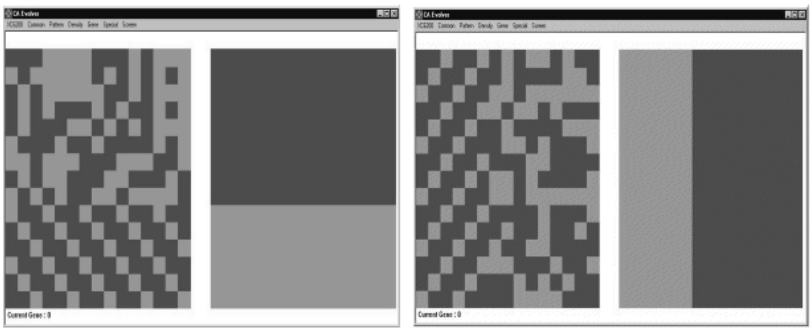


Fig. 4. (a) Horizontal and (b) Vertically segmented: input image left/ideal image right

is filled with noise of a density randomly selected from a uniform distribution between 0 and 1.

The state of each CA cell is first initialised with the corresponding point in the 15 by 15 training image. The XC6216 is then clocked for 200 cycles at 33Mhz to iterate the CA array. The fitness of the CA individual is calculated by a bit by bit comparison of the state of CA cells and the ideal image. A fitness counter is incremented for each CA cell in correspondence with the ideal image. While the counter is greater than zero it was also decremented for each cell that is not in correspondence. As in [4] CA individuals that lead to collapsed images (the state of the CA array ends up all 1's or all 0's) are penalised and receive a fitness score of 0. Each individual within the population is evaluated against 300 training images at each generation. As in [4] the overall fitness for an individual is calculated as the root mean square over the 300 images.

6 Performance Comparison

The computation times for the XC6216 based genetic algorithms were estimated from the CA Evolver experiments. A software based experiment that searched the XC6216 *hybrid* CA rule table was also implemented and is compared in Table 1. The computation time for the genetic algorithm can be calculated as:

$\mathbf{T_{total}} = \mathbf{T_{pop}} + Generations * (\mathbf{T_{fit}} + \mathbf{T_{ga}})$ where $\mathbf{T_{pop}}$ is the time to generate a population, $\mathbf{T_{fit}}$ is the time needed to evaluate the fitness of a population and $\mathbf{T_{ga}}$ is the time to generate a new population with the GA operators. Measurements were based on non-optimised code and it is expected that execution time could be significantly reduced. The relative measurement of speedup can be considered accurate and is summarised in Table 1.

We also make a qualitative comparison of the two FPGA based GA, *FPGA Bitstring* and *Logic Tree* representations as well as the software based rule table GA. For each experiment run the GA had 100 generations in which to find a CA to correctly classify the diagonal line pattern. Figure 5 illustrates the average pixel error as a function of GA time (generations). The *Logic Tree* representation

Measurement in Seconds	Software Rule table	FPGA Bitstring	Logic Tree	Relative Speedup
T_{pop}	0.1	0.1	0.1	1
T_{fit} per generation	1987.9	34.22	34.17	58
T_{ga} per generation	0.05	0.06	0.08	1

Table 1. Estimation of Speedup

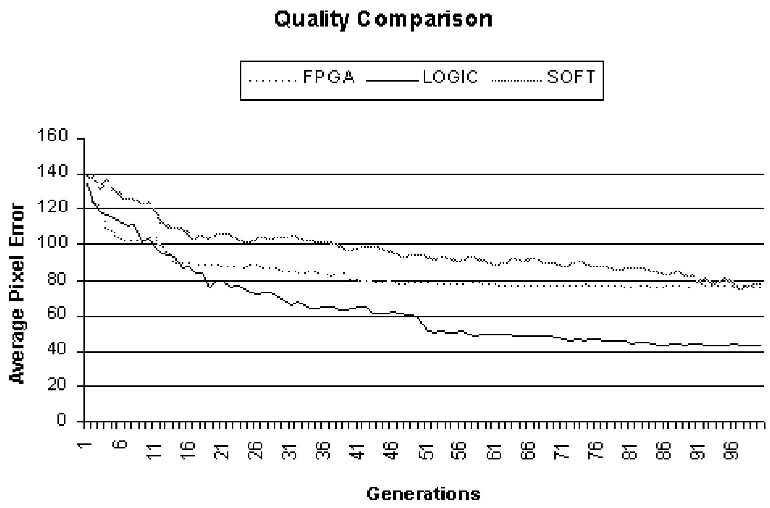


Fig. 5. Average pixel error in experiments

outperformed the *FPGA Bitstring* representation in all experiments. We conclude that constraining connectivity and evolving only function avoids strange feedback loop/analog circuit behaviours and allows a more detailed exploration of a smaller, more relevant rule space.

In several runs individuals of note appeared that performed well for images segmented in one direction (eg. vertically) but poorly in the other direction. To encourage the GA to find solutions in both directions the *Logic Tree* search space was further constrained. The multiplexer selector of node 4 in Figure 2(a) was pre-defined as the cell's current state in order to reduce the size of the non-symmetric rule space. Due to the close spatial relationship between training and ideal images, two nodes of type 1 (one in each half of the logic tree) were also pre-defined to store the initial training image data. The best of run CA from these experiments had an average pixel error of 40 pixels, lower than any other CA found. We conclude the *Logic Tree* GA can implement problem specific constraints more easily than raw FPGA bitstring and rule table based approaches.

7 Summary and Future Directions

The CA Evolver is a powerful tool with which to investigate genetic algorithm 'programming' techniques of cellular automata architectures. When searching XC6216 *hybrid* CA rule spaces speedup in the order of 10-100 times can be achieved compared to software implementations. Our initial experiments applied this tool to a binary texture analysis problem and investigated the role of representation. The FPGA based search space allowed problem specific constraints to be implemented easily leading to improved performance of the genetic algorithm.

Questions yet to be addressed include the role of non local CA neighbourhoods, extension to practical application and issues of scale. Eventually our work hopes to redefine these questions in terms of how best to 'program' XC6216 architectures as massively parallel machines and investigate a range of cellular automata models including random boolean networks.

8 Acknowledgment

This work was carried out in the Cooperative Research Centre for Satellite Systems with financial support from the Commonwealth of Australia through the CRC program.

References

1. J. E. Hopcroft and J. D. Ullman, "Introduction to Automata Theory, Languages and Computation", Addison-Wesley, 1979, Reading, Massachusetts.
2. F. C. Richards, T. P. Meyer, N. H. Packard, "Extracting Cellular Automaton Rules Directly from Experimental Data", Cellular Automata: Theory and Experiment, H. Gutowitz, ed., 1st MIT Press ed., 1991, Massachusetts.
3. M. Mitchell, J. P. Crutchfield and R. Das, "Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work", First International Conference on Evolutionary Computation and Its Applications (EvCA'96), Moscow, Russia.
4. P. Sahota, M. F. Daemi, D. G. Elliman, "Training Genetically Evolving Cellular Automata for Image Processing", International Symposium on Speech, Image Processing and Neural Networks, 13-16 April, 1994, Hong Kong.
5. J. R. Koza, F. H. Bennett III, J. L. Hutchings, S. L. Bade, M. A. Keane, D. Andre "Rapid Reconfigurable Field-Programmable Gate Arrays for Accelerating Fitness Evaluation in Genetic Programming", Late Breaking Papers at the Genetic Programming 1997 Conference, J. R. Koza, ed., pp. 121-131, 1997, Stanford.
6. A. Rosenfeld, "Parallel Image Processing Using Cellular Arrays", Computer, No. 16, pp. 14-20, 1983.
7. Xilinx Inc. "XC6200 Field Programmable Gate Arrays" Product Description, Version 1.10, April 24, 1997.
8. E.F. Codd, "Cellula Automata", Academic Press, 1968, New York.
9. E. R. Banks, "Universality in Cellular Automata", IEEE 11th Annual Symposium on Switching and Automata Theory, pp. 194-215, 1970, Santa Monica, California.