LAYERED APPROACH TO INTRINSIC EVOLVABLE HARDWARE USING DIRECT BITSTREAM MANIPULATION OF VIRTEX II PRO DEVICES

Rashad S. Oreifej, Rawad N. Al-Haddad, Heng Tan and Ronald F. DeMara

School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362
E-mail: demara@mail.ucf.edu

ABSTRACT

An integrated platform for fast genetic operators is presented to support intrinsic evolution on Xilinx Virtex II Pro Field Programmable Gate Arrays (FPGAs). Dynamic bitstream compilation is achieved by directly manipulating the bitstream using a layered design. Experimental results on a case study have shown that a full design as well as a full repair is achievable using this platform with an average time of 0.4 microseconds to perform the genetic mutation, 0.7 microseconds to perform the genetic crossover, and 5.6 milliseconds for one input pattern intrinsic evaluation. This represents a performance advantage of three orders of magnitude over JBITS and more than seven orders of magnitude over the Xilinx design tool driven flow for realizing intrinsic genetic operators on a Virtex II Pro device.

1. INTRODUCTION

Intrinsic evolutionary approaches such as Genetic Algorithms (*GAs*) are used throughout the literature to realize hardware-in-the-loop FPGA-based system design and repair strategies [1-3]. They realize search algorithms based on the Darwinian's evolution principles by performing genetic operations such as *mutation* and *crossover*. Many variations of GAs were introduced to enhance the performance and speed of convergence to a solution for FPGA-based systems [4], however, many of these algorithm implementations are software-in-the-loop simulations rather than real implementations on the FPGA fabric. Challenges of realizing practical intrinsic evolutionary strategies include the mapping of the genotype in the GA into its corresponding phenotype on the fabric, and the limited control over process automation of altering and downloading safe bitstreams onto the device. These issues are exacerbated when the critical portions of bitstream representation are proprietary.

In this paper, an approach that provides a fast interface between the GA and the FPGA device via a straightforward data-structure and Application Programming Interfaces (*APIs*) is presented. A layered design is used to perform mapping operations directly on the bitstream to modify LookUp Table (LUT) configurations, and reprogram the device. In addition, it supports Inputs/Output transfers via the JTAG standard serial port for fitness measurement purposes.

The remainder of the paper is organized as follows: Section 2 provides an overview of related work. Section 3 introduces the platform design. Section 4 discusses the experimental design and results, and Section 5 concludes the paper and suggests a direction for future work.

2. RELATED WORK

There are two paradigms for implementing GAs in reconfigurable applications: *Extrinsic Evolution* via functional models that abstract the physical aspects of the real device, and *Intrinsic Evolution* on the actual devices. It is evident that extrinsic approaches simplify the evolution process as they operate on software models of the FPGAs.

However for applications like in-situ fault handling on deep space missions, not all fault types can be readily accommodated by software models. Additionally, abstracting the physical aspects of the target device complicates rendering the final designs into actual on-board circuits, for instance, limitations such as routability of the design cannot be ensured until the final stages of the configuration process. For these reasons, intrinsic evolution can provide a direct approach to realizing physical designs for a specific FPGA device.

Several previous research efforts have addressed intrinsic evolution. A successful attempt on Field Programmable Transistor Array (FPTA) chips was carried out by [3]. They proposed new ideas for long-term hardware reliability using evolvable hardware techniques via an evolutionary design tool (EHWPack) that facilitates intrinsic evolution by incorporating PGAPack genetic engine with Labview test-bed running on UNIX workstation. They were able to intrinsically evolve a Digital XNOR Gate on two connected FPTA boards. In this paper,

we target FPGAs rather than FPTAs and namely the popular Xilinx Virtex II Pro device.

Miller, Thomson, and Fogarty [2] previously addressed the importance of direct evolution on the Xilinx 6216 FPGA devices; the research explored the effect of the device physical constraints on evolving digital circuits. A mapping between the representation genotype and the device phenotype was proposed, however, no implementation details were presented.

Hollingworth, Smith, and Tyrrell develop intrinsic evolution platform for a 2-bit adder on a Xilinx FPGA with partial reconfiguration to improve evolution time [6]. However, they used the JBits interface for run-time reconfiguration. JBits is Java-based, and being that it isinterpreted can face scalability and performance issues.

In a previous work, a *Multilayer Runtime Reconfiguration Architecture (MRRA)* was developed for Autonomous Runtime Partial Reconfiguration of FPGA devices [7]. The tool comprises three layers (*Logic*, *Translation*, and *Reconfiguration* layers) with well-defined interfaces for modularity and reuse. In addition, a standard set of Application Programming Interface (API) was utilized for communication with the target device. Results had shown the ability of the framework to support autonomous and dynamic reconfiguration operations. In this paper, MRRA is extended to support genetic operators directly to realize intrinsic evolution on Xilinx Virtex II Pro devices as discussed in the following sections.

3. JTAG-DRIVEN PLATFORM

The developed platform consists of hardware components that reside on the FPGA chip and software components which reside on the host PC, however, they are developed into layered modules that can be readily migrated to work on the PowerPC on chip in later phases of this research. The main components of the platform are shown in Fig. 1 containing components as follows:

I. JTAG Port: Standard JTAG (IEEE 1149.1) serial port. Its circuitry is implemented on the non-reconfigurable area at the top right corner of the Xilinx Virtex II Pro device and is embedded in most of the Xilinx Virtex and Spartan device families. This port provides a half-duplex serial communication interface.

In the developed platform, JTAG is interfaced via the *General-purpose Native jtAg Tester (GNAT)* [8] platform from the FPGA side, and to the parallel port (IEEE 1284) on the host PC using Xilinx Parallel Cable for input/output data exchanged between the host PC and the FPGA.

II. GNAT: General-purpose Native jtAg Tester component which has been developed as part of the bitstream on the reconfigurable area of the chip. It connects to the BSCAN_VIRTEX2 block via the `TDI`, `TDO`, and Control

signals, and to the targeted circuit via a simple read/write bus interface [8].
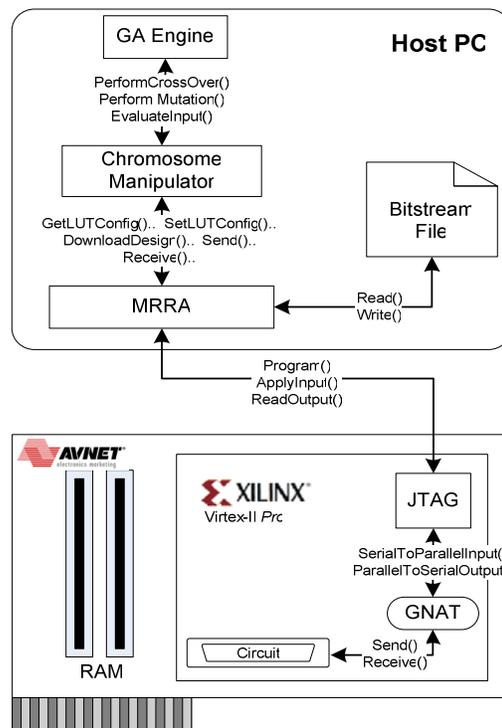

Fig. 1. Intrinsic Evolution Platform

III. Evolved Circuit: This is the subject circuit to be evolved on the FPGA chip. The circuit peripherals are connected to the read/write bus of the GNAT to receive input signals and confer the corresponding output signals.

The software components shown in Fig. 1 are as follows:

I. GA Engine: A C++ based console application implemented using an object oriented architecture. It contains classes which model the GA such as *Individual* and *Generation* classes along with the GA parameters such as the *Mutation*, *Crossover*, and *Elitism* rate. This module implements the conventional GA and is an independent component which can be replaced by any other enhanced algorithm variations. A conventional population-based GA was selected to demonstrate the applicability of the intrinsic genetic operators on the actual hardware. The handshaking between the GA Engine module and the Chromosome Manipulator module is done through a common data-structure that holds the genotype representation of the genetic individual.

II. Chromosome Manipulator: This is a C based library that contains the following functions:

a. *GetConfiguration*: Populates genotype data-structure from the configuration bitstream via the MRRA.

b. *PerformCrossover*: Performs a probability-driven single point genetic crossover on the two parent chromosomes. Crossover point is randomly assigned for both parents.

c. *PerformMutation*: Performs a probability-driven single-bit genetic mutation. One randomly selected bit of the binary chromosome content is flipped.

d. *ConfigureIndividual*: Maps back the chromosome's genotype into its corresponding phenotype and programs the FPGA via the JTAG port.

e. *EvaluateInput*: Applies test patterns to the circuit on chip via the GNAT module and reads the corresponding output.

In summary, this layer provides a logical abstraction of genetic operators to the GA Engine module. This facilitates the integration of any GA at the top layer by making the hardware implementation details transparent.

III. MRRA: developed by our team for autonomous runtime partial reconfiguration of FPGA devices [7]. MRRA operations are partitioned into a *Logic, Translation*, and *Reconfiguration* layers along with a standardized set of APIs. At each level, resource details are encapsulated and managed for efficiency, portability, and dynamic operation. In particular, FPGA configurations can be manipulated at runtime using on-chip resources on Xilinx Virtex II Pro platform.

IV. Bitstream File: A pre-compiled bitstream is generated beforehand using the Xilinx CAD tools. It contains the interconnected LUTs to be configured by the platform to evolve and realize an original circuit *Design* or restore functionality via *Repair*. The platform then manipulates this bitstream file to carry out the physical mapping of the crossover or mutation.

The workflow of the platform is divided into three phases:

1) Initialization:
 a) The baseline bitstream is manually designed using the Xilinx CAD tools.
 b) The GA requests the genotype representation of the baseline configuration from the Chromosome Manipulator layer.
 c) The Chromosome Manipulator module requests the chromosome LUTs configuration information from the bitstream file via the MRRA module.
 d) The MRRA module directly accesses the bitstream file and extracts the configuration information from the column-based vertical configuration frames using the *Frame based Partial Reconfiguration Flow* [7].

 e) The Chromosome Manipulator layer restructures the bitstream data into the genotype data-structure and sends it back to the GA Engine module.

2) GA Operations:
 a) The GA Engine calls the *PerformCrossover* or *PerformMutation* from the Chromosome Manipulator layer and passes the target chromosome(s).
 b) The Chromosome Manipulator performs the Crossover or Mutation and sends the produced offspring or the mutated chromosome to the Engine.

3) Fitness Evaluation:
 a) Chromosome Manipulator issues a download command to the MRRA which writes-back the individual's physical representation to the bitstream file. Then the file is downloaded to the FPGA via the JTAG port.
 b) Input patterns are sent serially in tandem to the FPGA via the JTAG according to the JTAG clock frequency.
 c) GNAT groups back the serial bits of each input pattern in the User Register and applies them to the corresponding circuit's input ports.
 d) As the output is evaluated, GNAT sends it to the MRRA which then passes it to the GA via the Chromosome Manipulator layer.

The genetic operations in the developed platform only take 0.401 microseconds for mutation and 0.709 microseconds for crossover. Hence, the only performance bottleneck is the communication speed with the FPGA chip. In this paper the JTAG serial port is used which imposes a substantial time delay that reaches up to 22 seconds to configure the entire device. This performance overhead can be considerably reduced if other interfaces are used such as the SelectMap parallel port or the *Internal Configuration Access Port* (ICAP) on a System on a Chip implementation using the PowerPC.

4. INTRINSIC EVOLUTION CASE STUDY

4.1. Experimental Design

The circuit used to demonstrate the platform workflow is a 4-bit x 4-bit adder. It provides a tractable circuit for the GA to evolve that exhibits characteristics for large arithmetic circuits including a variable amount of redundancy and combinational logic behavior. The circuit layout on Xilinx Virtex II Pro chip is shown in Fig. 2.

The GA parameters used throughout the experiments are shown in Table 1. Total of 8 LUTs were used in the design experiments, this number was increased to 13 LUTs in the repair experiment to add some redundancy margin for the GA to evolve within. All GA parameters were extracted by running extrinsic evolution of the GA and finding out the optimal values. Table 1 shows the range of tested values for each parameter along with the optimal one. Population

sizes between 5 and 20 were evaluated and best results were achieved using population size of 10. Crossover rates in the range 30%-90% (increment of 10%) were tested, the GA performed better when the value was set to 60%. Same applies to the other parameters.
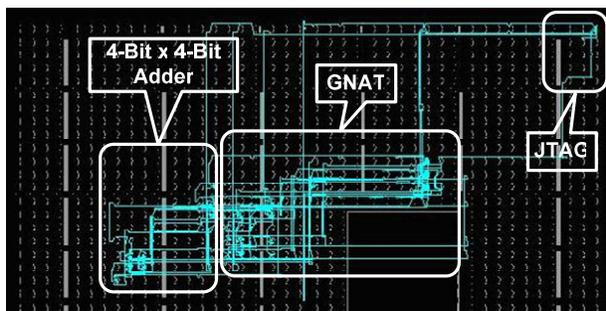

Fig. 2. 4-bit x 4-bit Adder, GNAT, and JTAG on Xilinx Virtex II Pro

Table 1. GA Parameters

Parameter	Range Evaluated	Value Selected
Number of LUTs for design	8	8
Number of LUTs for repair	8-13	13
Population Size	5-20	10
Mutation Rate	5%-90%	50%
Crossover Rate	30%-90%	60%
Tournament Size	1-8	6
Elitism Size	1-2	1

There are three types of experiments performed as follows:

Unseeded Design: In this experiment the GA evolved the 4-bit x 4-bit adder with a randomly-seeded initial population. The purpose of this experiment is to demonstrate the capability to intrinsically evolve 100% functional circuits starting from random bitstream. A baseline bitstream that contains 8 interconnected LUTs along with the GNAT core connected to the JTAG component was generated manually using Xilinx ISE.

Seeded Design: In this experiment, the GA evolved the 4-bit x 4-bit adder from an initial population of partially functional individuals in addition to completely random ones. The partially functional seeds were originally fully functional designs which were tampered by deliberately exposing them to mutation operator. This arrangement emulates a fault-scenario in real life avionics or space applications in which the configuration bitstream is partially affected by Single Event Upset (SEU) due to cosmic radiation. Typically, scrubbing is used to replace bitstream with an intact version stored on nonvolatile storage. However, this experiment could operate even in the event of local permanent damage to the underlying fabric even beyond SEUs.

Repair: A single stuck-at fault was adopted as a case study to show the capability of the platform to repair the faulty circuit. Since an actual fault cannot be readily nor precisely introduced into the device, the circuit is stimulated to behave as if the fault actually exists. This course of action becomes more complicated considering the fact that the platform allows only functional logic manipulation without the possibility of altering the device interconnects. Hence, the bitstream was processed directly before configuring the device to modify the contents of one LUT so that it behaves as if a stuck-at fault is present.

The LUT in Virtex II Pro chip is a 16-bit lookup table with four inputs and one output. If the Least Significant Bit (LSB) input pin is stuck-at zero, only the memory locations of the pattern $(XXX0)_2$ – where X is the *Don't Care* logic - will be accessible. This behavior can be achieved by copying the content of the memory locations of the pattern $(XXX0)_2$ into $(XXX1)_2$ and overwrites their old values as shown in Fig. 3.
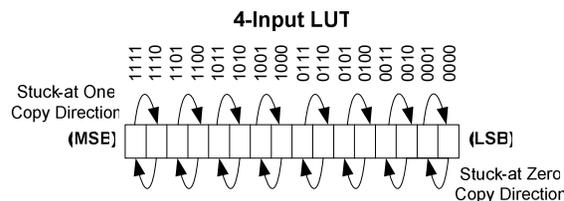

Fig. 3. LSB Stuck-at Zero and One Fault Modeling

Likewise, if the fault is stuck-at one in the second LSB input pin, and by following the previous analysis, any reference to $(XX0X)_2$ should be directed to $(XX1X)_2$. The same concept can be extended where the location of the error determines the stride between the memory locations to copy, and the value of the stuck at condition (zero or one) determines the direction of the copy operation (left or right) as shown in Fig. 3.

4.2. Results:

Five intrinsic evolutions were achieved for each of the unseeded, seeded, and repair experiments using the presented platform. The GA parameters listed in Table 1 were used. The following aspects were measured to quantify the capability of the platform:

a) F_{max} : The numerical measure of the fitness for the best individual of the final generation of the run. The maximum fitness for the 4-bit x 4-bit Adder is calculated as follows:

$$Fitness_{Max} = (\text{No. of Input Patterns}) * (\text{Output Width})$$
$$= (2^8) * (5) = 1280.$$

b) $\overline{F_{final}}$: The arithmetic mean for the fitness of all the individuals in the final generation of the run.
c) G : The total number of generations in the run.
d) **Timing Information**: The timing information for each run and is divided into the followings:

- CM_{total} : The time elapsed to perform the GA crossover and mutation during the entire run.

- $F_{evaluation}$: The time elapsed to apply the input patterns and read back the corresponding outputs for all the fitness evaluations during the entire run.

- $\overline{C}$: The average time taken by a single genetic crossover for a certain GA run.

- $\overline{M}$: The average time taken by a single genetic mutation for a certain GA run.
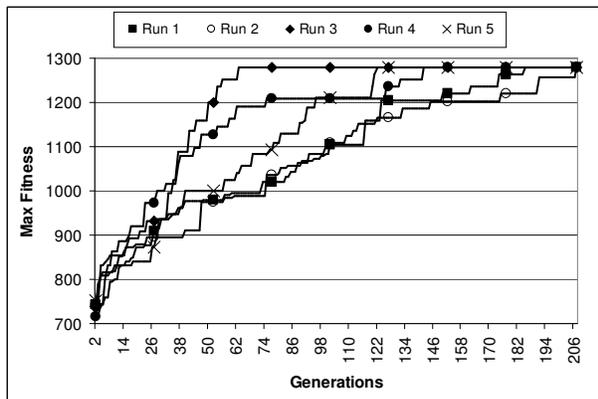

Fig. 4. Unseeded Design GA Runs

Experimental results are listed in Table 2. It can be seen from the results that the GA operators' time is small compared to the fitness measurement time. Moreover, it gets very small compared to the device programming time which was found to be 22 seconds. Device programming time is high due to two reasons: First, the JTAG serial port which can work at 300Kbps [9] was used rather than SelectMap interface that can operate at a maximum of 66MHz clock speed [10], second, 548Kbyte full bitstream file was used rather than the 80Kbyte partial reconfiguration file.

Fig. 4 shows five runs that demonstrate the capability of the platform to evolve to fully working 4-bit x 4-bit Adder designs starting from scratch. The maximum fitness starts as low as 716 out-of-1280, and rapidly increases during the first few generations

Fig. 5 shows five runs where a fully working 4-Bit x 4-Bit Adder was designed from a partially working seed. Five different seeds were used in the five runs.
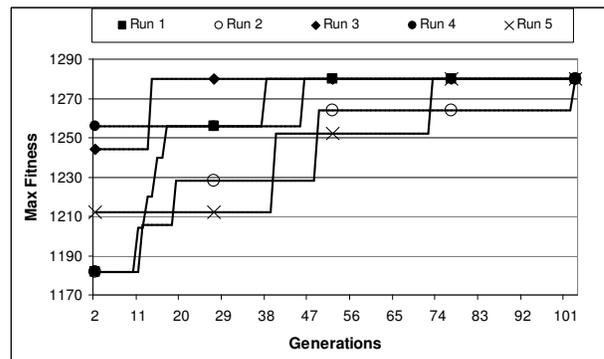

Fig. 5. Seeded Design GA Runs

Fig. 6 shows five runs in which the platform was used to repair the broken 4-Bit x 4-Bit Adder. A stuck-at zero fault was randomly injected in the first input pin of the third LUT of the original design. The fault was introduced using the technique mentioned in section 4.1. This fault reduces the circuit's fitness to 1152 out-of-1280. The fastest run was (Run 4) which reached to full fitness after 94 generations.

In Table 2, the timing measurement of the probability-driven mutation and crossover operators for each run is listed. The mutation and crossover average times throughout the runs were around 0.2 and 0.4 microseconds respectively. To measure the exact time that mutation and crossover operations require, another experiment was carried out by setting the mutation and crossover rates to 100% to ensure that the operators are performed with certainty. This allowed measurement of the time for each operation individually. The results of this experiment and similar experiments using Xilinx design tool driven flow and using JBITs are listed in Table 3. It can be seen from the results that more than seven orders of magnitude enhancement over Xilinx design tool driven flow and three orders of magnitude enhancement over JBITs was achieved by the developed platform.
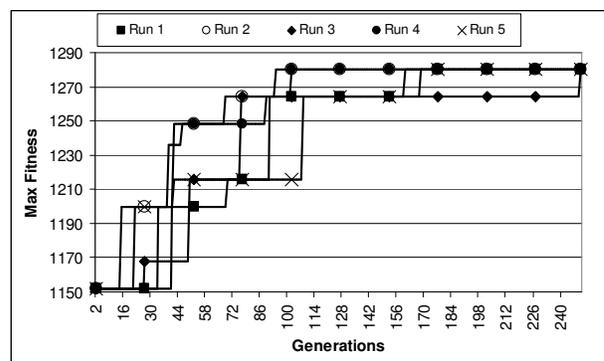

Fig. 6. Repair GA Runs

Table 3. GA Operators Timing (seconds)

This Platform		Xilinx Tool Flow		JBITS	
$\overline{C}$	$\overline{M}$	$\overline{C}$	$\overline{M}$	$\overline{C}$	$\overline{M}$
7×10^{-7}	4×10^{-7}	12.56	9.9	4.8×10^{-3}	4.6×10^{-3}

5. CONCLUSION AND FUTURE WORK

An intrinsic evolution platform is developed for genetic operators and fitness assessment using API layers which directly manipulate the configuration bitstream on Xilinx Virtex II Pro devices. Communication between the host PC and the FPGA device is carried out via the JTAG port. GNAT is utilized for intrinsic fitness measurement. Three experiments were conducted: unseeded design, seeded design, and repair. Experimental results have shown successful evolution with an average time of 0.4 microseconds to perform the genetic mutation, 0.7 microseconds to perform the genetic crossover, and 5.6 milliseconds for one input pattern intrinsic evaluation. Future work is proceeding towards a System-on-Chip version using the PowerPC to execute the genetic algorithm. This will reduce the significance of the data transfer time relative to genetic operator time.

6. REFERENCES

[1] S. Vigander, "Evolutionary Fault Repair in Space Applications," in *Dept. of Computer & Information Science*, vol. Masters Thesis. Trondheim: Norwegian University of Science and Technology (NTNU), 2001.

[2] J. F. Miller, P. Thomson, and T. Fogarty., "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study," in *Algorithms and Evolution Strategy in Engineering and Computer Science*, D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, Eds. Chichester, England, 1998, pp. 105-131.

[3] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica, "Fault-Tolerant Evolvable Hardware Using Field-Programmable Transistor Arrays," *IEEE Transactions On Reliability*, vol. 49, issue 3, September 2000.

[4] R. S. Oreifej, C. A. Sharma, and R. F. DeMara, "Expediting GA-Based Evolution Using Group Testing Techniques for Reconfigurable Hardware," in proc. *International Conference on Reconfigurable Computing and FPGAs (Reconfig'06)*, San Luis Potosi, Mexico, September 20-22, 2006, pp. 106-113.

[5] R. F. DeMara and K. Zhang., "Autonomous FPGA Fault Handling through Competitive Runtime Reconfiguration," in Proc. *of the NASA/DoD Conference on Evolvable Hardware (EH'05)*, Washington D.C., U.S.A, June 29-01, 2005.

[6] G. Hollingworth, S. Smith, and A. Tyrrell, "The intrinsic evolution of virtex devices through internet reconfigurable logic," in Proc. *of the Third International Conference on Evolvable System*, April 2000.

[7] H. Tan and R. F. DeMara, "A Device-Controlled Dynamic Configuration Framework Supporting Heterogeneous Resource Management," in proc. *of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'05)*, Las Vegas, Nevada, U.S.A, June 27-30, 2005.

[8] D. Wallace, "Using the JTAG Interface as a General-Purpose Communication Port," www.xilinx.com/publications/xcellonline/xcell_53/xc_pdf/xc_jtag53.pdf, 2005.

[9] Xilinx, "Parallel Cable IV Connects Faster and Better," *Xcell Journal*, Spring 2002.

[10] Xilinx, "Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode," v1.4 November 13.