Research on the Architecture of Intrinsic Evolvable Digital Circuits

NIE Xin, LI Yuan-xiang, KE Peng
State Key Laboratory of Software Engineering, Wuhan University, Wuhan, China
nix83@163.com
doi:10.4156/jnit.vol2.issue2.2

Abstract
At present, the main research in Evolvable Hardware (EHW) field is focused on the intrinsic evolution based on the virtual reconfigurable circuit. On the basis of previous researches, we take a further explore on designing a virtual reconfigurable architecture for intrinsic evolvable hardware, and propose a virtual circuit architecture based on Look-Up-Table, each evolution module of which can be configured more functionality and more connectively. Then a parallel fitness evaluation module will be introduced for fast fitness computation – normally the most time consuming part of evolution. This intrinsic evolution architecture is implemented in a Xilinx Virtex-II Pro FPGA (XC2VP30) with an embedded PowerPC processor which allows for the processing of hardware software co-evolution. The experimental results show that the new architecture can bring higher evolutionary ability and more generality and flexibility than traditional approach to intrinsic EHW.

Keywords: Evolvable Hardware, Intrinsic Evolution, Virtual Reconfigurable Circuit, Look-Up-Table

1. Introduction
Evolutionary algorithms (EA) [1] are stochastic search methods that mimic the metaphor of natural biological evolution. Evolutionary algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. Impressed by Charles Darwin, Prof. John Holland of the University of Michigan viewed the process of biological evolution as a process of optimization, where nature selects the best evolutionary settings to survive in the next generation of offspring [2]. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation.

Evolutionary algorithms model natural processes, such as selection, recombination, mutation, migration, locality and neighborhood. Figure 1 shows the structure of a simple evolutionary algorithm. Evolutionary algorithms work on populations of individuals instead of single solutions. In this way the search is performed in a parallel manner.

![Figure 1. The evolutionary algorithm operators](image)

The process of developing an EA for a particular application consists of the following chief phases:
2. Evolvable Hardware

Inspired by natural evolution, evolvable hardware (EHW) [3] was developed in the early 1990s as a new concept in the development of adaptive machines. In contrast to the traditional hardware where structure and functions are irreversibly fixed once implemented, EHW refers to a type of hardware whose architecture and functions can change dynamically and autonomously by interacting with its environment. This adaptation ability of EHW, achieved by employing evolutionary learning and reconfigurable device, has great potential for the development of innovative and powerful real-world applications [4].

The basic idea of Evolvable Hardware (EHW) is to regard the configuration bits of a software-reconfigurable device as the chromosome of EA. The search-space of configurable bits is very huge, but EAs are very effective without prior knowledge of the search-space. As a fitness function, we choose the performance of the hardware circuit. For example, in Data Compression with EHW, we use a predictive function implemented with hardware. As a fitness function we choose the data compression rate. When a good chromosome is obtained, it is immediately downloaded into the reconfigurable device.

In EHW, it is not required to specify the detailed hardware design. Instead we define a fitness function. A fitness function is the instinct of the circuit to evolve itself. If the fitness value of a hardware circuit is degraded due to partial malfunction or some changes in the environment, then the
EA-process of EHW is invoked, and the search for a better hardware configuration is initialized. Hence, EHW continues to reconfigure itself in order to get better performance.

The chromosome of EHW specifies two things. One is the function type of the evolution unit. In figure 3, the evolution units correspond to gates like AND-gate and OR-gate. The other is the interconnection among the evolution units. EHW can be classified into two classes according to the grain-size of an evolution unit; gate-level and function-level. The figure is an example of gate-level evolution. In function level evolution, each evolution unit is higher hardware function than gate-level evolution [5].

![Evolvable Hardware](image)

There are many methods to implement intrinsic evolution. A methodology for direct evolving is described in, it is the most intuitionistic intrinsic evolution, however, this approach requires a very professional understanding of a given FPGA as well as a familiarity of its configuration bitstream structure. Although the author introduces the bitstream composition of XC2V40 and teaches us how to localize the LUT contents in a configuration bitstream, there are two fatal limitations in its flexibility: On one hand, even produced by the same corporation, different types of FPGA have different configuration compositions, if the experimental environment is changed, we must spend unwanted energy on familiarizing ourselves with the new environment and re-parse the configuration, then locate the LUT contents from millions of configuration bitstream bits, and link them as a gene for evolution, that is obviously a lack of portability. On the other hand, illegal bitstream may destroy FPGA, however, evolve the configuration bitstream directly will generate some illegal bitstream easily, when these illegal ones are downloaded to the FPGA, the chip will not work even damaged.

### 3. Complete intrinsic evolution on FPGAs

#### 3.1. Virtual reconfigurable architecture

To speed up fitness evaluation, with the rapid development of reconfigurable devices, different intrinsic EHW approaches have been proposed in recent years. FPGAs which enjoy both the high performance of a dedicated hardware solution and the flexibility of software offered by its inherent reprogram ability feature are the most popularly utilized commercial reconfigurable logic device for digital intrinsic EHW. A number of works have been done in the area of FPGA-based intrinsic EHW.

In this paper, we develop and design a virtual reconfiguration-technique-based EHW platform, named virtual reconfigurable architecture (VRA) [6]. The VRA, which is described in an HDL, is a second reconfiguration layer developed on the top of an FPGA. The key goals of our proposed VRA are to provide a much simpler intrinsic EHW platform – thus reducing the length of the genotype description and providing the feature of fast internal reconfiguration.

A basic schematic of the configurable cell is shown in fig 4. It has a 8-bit input and a 1-bit output. The LUT (look up table), shown in right side, provides the logic operation. The three multiplexers,
shown in MUX1 MUX2 and MUX3 are used to select the inputs to the LUT. So, each LUT input can be driven from 1 of a set of 8 inputs.

Fig 4. The architecture of Configurable Cell

In total, this configurable cell requires 17 configuration bits. Each multiplexer requires 3 selection bits. The LUT has 8 selectable outputs. So the total number of configuration bits = (3 * 3) + 8 = 17 bits.

Fig.5 shows that our VRA consists of some function element (FE) arrays. The top function of the FE array is configured using the chromosomes generated by EA, which is implemented on the same FPGA. The chromosome encodes the functions performed by each FE and the interconnection of the FE array. As the fitness calculation is also carried out in the same FPGA, we can benefit from pipeline processing allowing reasonable time of a candidate circuit evaluation.

The configurable circuit is created from an array of these cells in fig 4. The array is shown below. Each cell input is connected to the outputs of the one previous column. For the special case of the first column, the previous inputs are in fact connected to the cell array inputs.

Fig 5. The architecture of VRA

From this perspective, the approach utilizing a VRA offers many benefits, including: (1) In the VRA, the FE array is directly connected to the hardware implementation of the EA placed on the same FPGA allowing the bottleneck introduced by slow communication between FPGA and PC can be
overcome. (2) Because the VRA is available at the level of HDL source code, it can easily be modified and synthesized for various target platforms. (3) The VRA can be designed exactly according to the requirements of a given problem. This feature means that the granularity of the FE array can exactly fit the needs of a given application.

3.2. Genotype–phenotype mapping

A pivotal problem in EHW domain is how to encode circuits to chromosomes, which has a direct influence on both the course of evolution and the final outcome. A good coding should satisfy three conditions at least: the first, its length should be controlled in the range which the EA could handle; the second, it should be able to decode to a practical circuit conveniently; the third, the evolution of the coding should be able to reflect that of both functionality and routing. The proposed methodology in this paper has advantage in obtaining such a good coding.

In the experiment, we evolved 2 × 2 bit multiplier and design an 8*8 VRA, the modules in every column were 8 inputs numbered from ‘0’ to ‘7’, the anterior 8 sequence numbers represented the 8 primal inputs and the latter 8 sequence numbers represented the 8 inputs from the previous column’s outputs. The 8*8 VRA has 64 such modules, joint these 64*17 integers together as a chromosome, so the length of chromosome in our experiment is 1088.

4. Experiment

4.1. Implementation of VRA

In the proposed system, designing the virtual circuit in VHDL is a preliminary work. The virtual circuit contains the three modules: the first module marked “FPGA Space” is a location allocated from all usable memory resources mentioned in the previous section, the second is VRA, and the third is a Counter.

The FPGA Space module is prepared for receiving chromosome and returning the Truth Table, this module is an alternating interface between the EA process and the VRA: the EA process writes every chromosome to FPGA Space module, and the FPGA Space module returning the Truth Table to EA process. The VRA module read datum from FPGA Space module, and then maps them to a real function circuit. The Counter module’s outputs are the inputs of the VRA module, it starts generating the input combinations of the VRA module as soon as the circuit finished mapping, once an input combination has been generated, a corresponding output combination will be written to an appointed location of the FPGA Space module, when all the input combinations have been completed, these locations constitute the Truth Table of this circuit.

When this preliminary work is finished, it is a universal design for evolving different target circuits, or only needs some tiny modification such as broadening the size or shifting the routing of VRA to evolve more complex target circuits.

The cell array provides us with a method of testing candidate circuit solutions as part of an evolutionary algorithm. A major advantage of performing fitness evaluations in hardware is that performing multiple evaluations in parallel can be achieved quite simply. In fact 8 cell arrays have been combined into a single block called the EvoBlock. The EvoBlock also contains extra circuitry to help determine the fitness of an individual.

The figure to the left shows the basic parts of the EvoBlock Module. The block of RAM is used to store the truth table of the target circuit. When test vectors are applied to the EvoBlock the correct output value is output from the RAM. This can then be compared to the output from each cell array.

The direct output from each cell array can be read, but a match output, is also generated. This output shows which bits of the RAM output and the cell array outputs match, i.e. which bits are correct. The xnor gate outputs a ‘1’ on a match and a ‘0’ when the bits differ. To determine how many bits the current individual has correct for a particular input vector, you just need to count the number of 1s in the match output.

The final feature of the EvoBlock is the mask register. The mask register is used to mask out unwanted outputs from the match result. This is used when the circuit being evolved does not make use of all 8 cell array outputs. For example the 2-bit multiplier only has three outputs, so if these
signals map to cell array outputs dOut2 to dOut0, the mask register will be loaded with 0x7. The match values for bits 7 down to 3 will always be '0'.

4.2. Result

The fitness is evaluated by GA, once the GA process obtains the Truth Table of current circuit from the appointed locations of FPGA, it compares this truth table with the Truth Table of the target circuit, checks them bit by bit and counts the match-bits, the percentage of the match-bits is the fitness of current circuit. There will be some trivial differences in fitness evaluation between different GA processes.

Evolution runs were conducted on our on-chip evolution system and a Pentium 4 (P4) workstation for speed comparisons. The P4 workstation has a clock frequency of 2 GHz. For the speed test, 10,000 generations of 20 individuals were evolved. The fitness evaluation was for a 2 × 2 bit multiplier [7, 8, 9]. Correct 2 × 2 bit multiplier circuits were evolved after an average of 4902 generations over 10 evolution runs. The same experiment was conducted as verification on the PC platform, where the average was 5349. The different numbers can be explained by the different programs using different random number generators. Still, the results are rather similar, which indicates that the FPGA implementation works correctly.

**Table 1. Device utilization for the EHW module**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Used</th>
<th>Available</th>
<th>Present</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>1025</td>
<td>4928</td>
<td>20</td>
</tr>
<tr>
<td>Slice Flip Flops</td>
<td>869</td>
<td>9856</td>
<td>9</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>1231</td>
<td>9856</td>
<td>12</td>
</tr>
</tbody>
</table>

In our experiment, the max generation of GA was 2,000,000, and the program was executed 10 times, there were 9 times step into “fitness stalling effect”, only once the fitness reached 100%, the low success rate is mainly attributed to the following reasons:

Firstly, theoretically speaking, an 8*8 VRA is enough to evolve a 2-bit multiplier, but in fact, it is difficult to evolve a fully functionality 4-bit adder in finite generations than size of 8*8, so increasing the size of VRA will help to improve the success rate.

Secondly, the routing between multiplexer modules in our experiment was stiff, every module could only connect to the previous column modules or primary inputs, and each module has only one output, which result in many modules have not been fully utilized, so a more flexible routing will be beneficial to increasing the likelihood of success evolution in limited size.

![Fig 6. The EvoBlock Module Structure](image-url)
Thirdly, the function styles defined in the proposed system are too simple, see Table 1, all the function styles are only one output, extending multi-output function styles or more complex function styles will be able to improve the efficiency of evolution. In addition, the GA operators in our experiment were not efficient enough, because we did not try our best to seek a better algorithm for evolution, this maybe another reason for the low success rate.

5. Conclusion

We discussed in this paper, the application of Evolutionary Algorithms to the problem of developing Evolvable Hardware. Evolvable Hardware can deal with practical industrial applications, especially those that require the ability to cope with time-varying problems and real-time constraints.

The powerful computation ability of our proposed evolvable platform is presented in the experimental results. When compared with the equivalent software simulation, our FPGA implementation obtains a performance increase of over 100 times in all cases. On the other hand, according to the analysis of the evaluation process in intrinsic evolution, the proposed VRA shows a promise to avoid the bottleneck introduced by the slow reconfiguration speed in traditional FPGA configuration bitstream-based intrinsic EHW. Future work will be concentrated on developing the reported VRA for solving more complex and demanding real-world industrial problems.

To conclude, Evolutionary Algorithms is currently a fertile ground for research and application development. While a rich set of techniques and models are available, covering a range of domains, there are many areas remaining to be understood and exploited. It must however be noted, that this technique (as every other technique!) poses some limitations over the application areas it can be used in, and hence scope for further development and research in this area is vast.

6. References