Evolvable Hardware in Theory and Implementation

Kyler Hauschildt

ABSTRACT

This paper will go through the relatively short history of evolvable hardware. From there we will discuss present day applications of evolvable hardware and how well these solutions compare to traditionally obtained solutions. Then we will talk about the future of the technology and steps being taken to overcome the limitations of this new field. An implementation of evolvable hardware on a field programmable gate array (FPGA) using a generic evolutionary algorithm to step through iterations until a solution is obtained will also be discussed to give the reader a clearer understanding of what implementations do behind the scenes.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.6.3 [**Simulation AND Modelling**]: Applications

General Terms

Field Programmable Gate Array, Evolvable Hardware, Evolutionary Algorithm

Keywords

Multiplexer, Configurable Logic Block, Wires Under Test, Programmable Interconnect Point

1. BACKGROUND

Moore's law stated in 1965 that "The number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years." [8] This prediction has held up for the past 45 years. Many speculate how long this growth can be sustained. In addition to increasing transistor density several other solutions to the problem of computing power have been approached. A promising approach in recent years has been that of evolvable hardware. The term evolvable hardware (EHW) in its modern use was coined by Hugo de Garis in 1992 to describe work he was doing in using genetic algorithms to evolve neural networks [10]. At the time, EHW was not seen as having much application outside of the research field. Evolvable hardware began to be seen as a serious vehicle for solution development in 1996 when Adrian Thompson at the University of Sussex, England published an evolved tone discriminator, a device capable of distinguishing between frequencies, using fewer than 40 logic gates in a field programmable gate array (FPGA, a reprogrammable circuit with a grid of logic gates) [13]. This accomplishment renewed interest in FPGAs as an implementation of evolvable hardware and may be seen as the cause for the paradigm shift of FPGAs right before the turn of the century [11].

2. INTRODUCTION

2.1 Evolvable Hardware

Evolvable hardware is a field of design that uses evolutionary algorithms to train circuits to meet a design specification. Evolvable hardware pulls from other fields such as artificial intelligence, reconfigurable hardware, and mathematical optimization to design a pool of candidates and, through evolution, refine the candidate circuits to satisfy the design specification.

2.1.1 Applications

Evolvable hardware is often implemented with *field programmable gate arrays* (FPGAs). Since FPGAs make up a very large part of current day evolvable hardware, most of this paper will focus on implementing evolutionary algorithms on FPGAs. FPGAs are, simply put, reprogrammable integrated circuits. They are a likely medium for EHW because they are affordable, well documented, and easy to program for.

2.1.2 Benefits of EHW

As discussed earlier, evolvable hardware is an approach to solve problems without increasing the number of transistors on a circuit and instead tries to evolve a set (or field) of programmable logic gates. In addition to not relying on the sustained exponential growth in affordable transistor density, evolvable hardware explores exploits of evolved circuits not thought of by humans. Another benefit of using evolvable hardware to solve a problem is adaptability. Since new fitting solutions can be derived on the fly, the system can maintain integrity in a changing environment [1]. Also, evolved circuits are in many cases more compact and energy efficient than human created circuits. This phenomenon can be explained through a combination of natural compression and evolved "shortcuts" humans did not think to use.

2.2 Evolutionary Algorithms

Evolutionary algorithms are often used to evolve a circuit on evolvable hardware to obtain a circuit that meets or exceeds the required fitness. Evolvable hardware implements evolutionary algorithms on a piece of hardware. The characteristics available to the engineer are translated into genes which are evolved and refined until a working solution is found and implemented on physical hardware. For example, the genes of a system trying to evolve a robotic hand that excels at gripping may have genes for number of the fingers, the number of joints, and the position of joints in the hand. Evolutionary algorithms are a subset of evolutionary computation which tries to evolve a population based on mechanisms seen in biological evolution, namely: reproduction, mutation, cloning, and selection.

Reproduction is the passing of parent "genes" to offspring in the hopes that the offspring will inherit beneficial genes from both parents. *Mutation*, much like biological mutation, occurs when genes are randomly changed in hopes of introducing new properties into the next iteration. *Cloning* occurs when a (near-passing) set of genes is passed directly onto the next generation in hopes of crossing with the right mate to produce a candidate that passes the fitness function. Finally, *selection* is defined as selecting more fit candidates from the pool to pass on their genes to future generations. These mechanisms are implemented in a software and hardware environment. For this paper, it is enough to say that genetic algorithms belong to the superset of evolutionary algorithms and treat the evolvable hardware approach to solving problems as a set of evolutionary algorithm techniques implemented on hardware.

3. EVOLVABLE HARDWARE

Before any actual calculations are made, the engineer describes a fitness function, a function that assigns each candidate a fitness value relating to how well that candidate meets or exceeds the solution specification. If a member has a fitness level greater than the target fitness level the hardware considers the problem solved and returns the acceptable circuit (member) and quits iterating. It is important to note that an algorithm generates a large group of individuals, models evolution in iterations, runs each individual against the fitness function, and continues this until an individual is evolved with a set of genes that passes the fitness function. As stated earlier, fitness functions are not always easy to describe. For example, a fitness function for an antenna design may require the antenna to have a gain greater than 2 decibels or a fitness function for an integrated circuit may require the circuit to run on an input voltage of 3.0 volts [5]. There is difficulty not only in translating requirements into logic, but also in the fact that there are generally many properties that need to be assessed given only the information in the genes. Through the process of selective breeding, members pass on their beneficial genes as determined by the fitness function.

3.1 Intrinsic and Extrinsic Evolution

Post-tone-discriminator research has split evolvable hardware into two approaches: *intrinsic* and *extrinsic* evolution. In the former, each candidate is implemented on physical hardware, very often an FPGA [2]. In the latter, each candidate is simulated on reconfigurable circuits, often FPGAs. The intrinsic approach has the benefit of being very accurate with regards to adapting theoretical solution to actual solution. That is, once a solution is found (a candidate has a fitness greater than the desired fitness) there is almost nothing that can change from the circuit's implementation to the real world implementation. Another benefit of this approach is the ability to create self-evolvable hardware systems. Self evolvable hardware systems are able to rework their circuits in the field and do not rely on waiting for a final solution to be found (or not found) before evolving. The main drawback of intrinsic evolution is the significant overhead of having to reconfigure the physical circuit for each generation [11].

Extrinsic evolution has, understandably, the opposite set of benefits and drawbacks. Where intrinsic evolution takes a large amount of time to rework the physical circuit, extrinsic systems simply implement an evaluation and simulates the new circuit. The main drawbacks of extrinsic evolution are that the simulated circuit is not as easily mapped to a real world circuit and often is not as efficient as a solution from intrinsic evolution [2]. In a simulated environment components are linked with simulated interconnects. In the real world the interconnects have a size and location. Without physically implementing the circuit one may wind up with a solution that can not physically be implemented or a solution that needs to be adapted to fit the physical limitations of the implementation [11] [3]. In practice extrinsic evolution is more common to obtain a working or near-working solution which needs to be tweaked. This takes out the complexity of needing to build a physical candidate every time. Although extrinsic evolution is utilized more often, intrinsic evolution has its uses. One area where intrinsic evolution occurs is the construction of a circuit on an FPGA by constructing virtual components in the logic gates [7].

4. FPGA IMPLEMENTATION OF EHW

At the core of all evolvable hardware is code that passes down the positive traits of past iterations. There are other tactics besides the passing of beneficial traits, but trait-passing is at the heart of evolving a circuit to match a fitness function. What follows is a walk through of a hypothetical implementation of a genetic algorithm on an FPGA that uses tactics such as cloning and mutation in addition to trait-passing to evolve a circuit to meet a desired fitness. It is difficult to describe in specific terms how an FPGA works because there are several manufacturers who each have their own design of FPGAs [9]. We will, however, explore as much as possible while still not becoming application-specific in our descriptions.

4.1 Field Programmable Gate Arrays

In general, an FPGA contains a grid (or field) of configurable logic blocks (CLBs) which can be either single logic operations (AND, NOR, etc) or groups of logic blocks that perform higher level functions (adder, multiplier, etc) depending on the sophistication of the FPGA. Each CLB is connected to the grid via a switch matrix that allows or disallows a connection to be made via a programmable interconnect point (PIP). The system is fed input which runs through the logic blocks in a manner determined by the state of the PIPs and gives an output which is evaluated with the fitness function.

4.2 Inheritance in Algorithms

For most of EHW, especially that on FPGAs, implementation code is an evolutionary algorithm. Generally speaking, the hardware starts out by randomly generating a group of circuits to act as the first generation. Each circuit is either simulated (extrinsic) or constructed (intrinsic) and evaluated with the fitness function. It is important to note here that the members chosen are not always the most fit, but more fit members are weighted heavier to pass on their genes than less fit members, hence the parallel to real world evolution. The processor of the FPGA weights each member based on fitness of genes.

Two selected members will combine their complementing genes in what is normally called a *crossover*. Aside from direct crossovers which take genes solely from the parents, there are also randomly occurring mutations - again, just like real world evolution. Mutations serve the function of introducing new and possibly beneficial genes into the candidates' gene pool. In this way, a gene that was erroneously eliminated earlier can be introduced into the population again. In an FPGA implementation adding or subtracting chromosomes is done by opening or closing PIPs [6]. Occasionally cloning will take place. Cloning in this sense is when a member, instead of mating with another member and producing an offspring of mixed traits, passes its full set of traits to the offspring, often only the case when a member has a very high fitness value.

4.3 Generation Cycling

When the number of offspring reaches the number of parents, a generation cycle occurs. Described in more detail in section 5.2, generally speaking, a generation cycle is when the offspring (group of just-generated candidates) become the parents for a new generation. Generation cycling is possible because each generation is stateless. That is, once a generation is generated it no longer requires the past generation so the past generation can be overwritten by a new generation [1]. A general generation cycle is shown in Figure 1. Typically, a given problem requires a number of generations in the tens or hundreds until a suitable candidate is discovered. The number of generations depends on many factors including how difficult it is to exceed a fitness bar given the requirements of the problem and the number of parameters each candidate must evolve. For example, the antenna design below took around 100 generations, with the best candidate emerging around generation 60 with the following 40 generations failing to find a better solution [5].

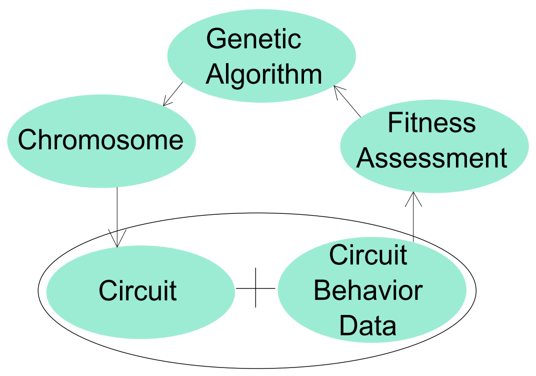

Figure 1: The life cycle of an evolutionary algorithm (here a genetic algorithm) that continually creates new circuits to test against a fitness function. [11]

5. DIFFICULTIES AND LIMITATIONS OF FPGA BASED EHW

5.1 Defining Parameters

Often it is hard to describe a desired parameter and restrictions in mathematical terms. For example, it is difficult to describe, in an algorithm, a 3-dimensional shape that needs to consist of a single wire of unlimited length but one that needs to fit inside a cubic foot box. Further complicating this is the representation of the output of an evolutionary algorithm. In the example just mentioned, the engineer would need to take the building instructions outputted by the FPGA and render them in 3-dimensional software to understand the candidate being proposed. An individual gene in this example may be expressed as an angle of a bend in the wire or of a length of straight line the wire requires [5]. Yet in another example, genes may refer to the delay in milliseconds the EHW waits to respond to input. Of course, both of these are hypothetical genes in hypothetical systems but they serve the purpose of showing how difficult it can be to specify a gene or limitation for EHW.

5.2 Scalability

There are several challenges inherent to evolvable hardware. Since one is evolving a random circuit in an unknown direction, that could be viewed as a "guided brute force" way to solve a problem. Evolvable hardware (that implements a genetic algorithm) relies on crossing genes of the fittest chromosomes to create a child generation. If a chromosome has 4 bits of genes it is easy to try all possible combinations of children each generation. If a chromosome has 8 bits of genes one would naturally select the best traits from each parent chromosome. If you encounter a problem where the parent chromosome has 32 bits of genes there is not an easy time/coverage trade-off you can make [13]. This can be stated another way. Evolvable hardware implementing genetic algorithms does not, by design, scale well. The more complex a "parent" is the harder it is to generate a representative population of children. The way an evolutionary algorithm works when applied to evolutionary hardware, the parent generation generates offspring until there are as many children as there are members in the parent generation. At that time the generations cycle; the children become the parent generation and the former parent generation are deleted to make room for the next generation of offspring. One can easily see that the number of offspring the algorithm generates is limited not by how many offspring make good candidates but by how many members the hardware can support. This can be expressed by the relationship Members Generated per generation = (1/2)(Member Capacity of Hardware) When you add in the exponential growth of mutations and cloning the coverage becomes unmanageable very quickly [7]. This weakness shows itself in the real world applications of EHW by the fact that EHW cannot, normally, solve complex problems in a timely manner, much like a systematic, evolving brute force attempt.

5.3 Interconnect Faults

In FPGAs there exists four basic types of interconnect faults. Interconnect faults are very small hardware faults involving a switchable data path being permanently stuck open or stuck closed. Since FPGAs contain an extremely large amount of interconnects, it is feasible for errors to occur. After listing the faults, we will discuss ways of detecting and mitigating these faults. Line segment faults are faults within an FPGA between programmable interconnect points (defined previously). The faults come in two states; open and closed. *Line segment open faults* are caused when line segments lose their ability to establish the connection between the pair of PIPs they join. Line segment open faults can be viewed as shorts between the power supply and ground, letting nothing flow between the pair of PIPs [6]. *Line segment closed faults* are caused when line segments maintain a permanent connection between their pair of PIPs, regardless of whether they are supposed to or not.

Programmable interconnect point faults are faults inside of the PIPs themselves. As discussed previously, PIPs have a series of connections inside them and are vulnerable to faults. PIPs also have two types of faults: *stuck-open faults* and *stuck-closed faults*. Stuck-open faults are a problem because a PIP will not be able to allow data through that point. Stuck-closed faults have the opposite problem; they will pass all information through that connection. The open/closed nomenclature is easier to understand if one thinks of PIPs as points in a circuit. Therefore, if a PIP is stuck-closed this would correlate to a *closed circuit* which would allow data to flow. Similarly, a stuck-open circuit would correlate to an open circuit where data cannot get from one point to another.

Faults in a system as large as an FPGA can easily be routed around and go unnoticed, at the cost of errors and diminished ability. In 2005 a method for finding faults, called Wires Under Test, was improved upon by the addition of buffers in the form of routing the testing signals through IMUX (input multiplexers, take several inputs choose the best one to relay), transparent logic, flip-flops, and OMUX (output multiplexers that choose which output to relay). One might wonder why buffers are necessary. Before buffers there were only unbuffered WUTs. Unbuffered WUTs had the disadvantage of limiting the number of tests that can be sent simultaneously to the number of I/O pins on the FPGA. The introduction of buffered WUTs allowed for the same coverage without the extensive amount of test configurations needed for unbuffered WUTs. This, when coupled with programs that generate test configurations, made finding faults in FPGAs cost and time effective. As an example, the number of test configurations needed for the FPGA in [6] went from sixty before interleaving (buffering) to just eight after.

6. APPLICATIONS OF EVOLVABLE HARDWARE

6.1 Appeal of Evolvable Hardware

Evolvable hardware has seen a love-hate relationship with the academic research community and is slowly being adopted as a legitimate (and not novel) way of solving problems. First, evolvable hardware had to be proven possible on consumer hardware. It found a home in the FPGA community where it was discovered that one could overlay a genetic algorithm over an FPGA and obtain cheap, effective evolvable hardware. Then, evolvable hardware had to jump the hurdle of having real-world applications. That was shown to be possible to a large extend in 1996 by Adrian Thompson with his sub-40 gate tone discriminator. Currently, evolvable hardware is being held back by scalability issues caused by complexity growth. Scalability with regard to complexity is essential for evolvable hardware to be considered in real world applications (many of which are prohibitively complex for evolvable hardware to be employed [13]).

6.2 Notable Real-World Uses of EHW

Despite the challenges, evolvable hardware has shown a glimpse of the ingenious solutions it is capable of evolving on par with, and surpassing human abilities in several notable examples. One area in which EHW excels is antenna design. Antennas are difficult to design because of the vast number of variables one must factor into his or her equation when trying to find an optimal design. For example, the angle that the signal will be coming in on, the allowed length of the antenna tines (or dish diameter), and the adjustment needs of the antenna are just some of the things to consider when trying to find an optimal antenna design.

Researchers at the NASA Ames Research Center (NASA ARC) had to address all of these limitations in 2004 when they were tasked with designing an antenna to be placed on NASA's then upcoming *Space Technologies 5* microsatellites. The team chose to develop the antennas using evolvable hardware. The initial results of this evolution are shown below in the *original evolved antenna* in Figure 2. A second round of evolution, this time adding the restriction of "no branching" to the requirements yielded the antenna shown in Figure 3. The result was a compact antenna design that would not have been considered in traditional research. It also took less time to evolve the antenna than it would have taken to design the antenna the traditional way. The evolved antenna took NASA approximately three person-months to set up and develop the algorithms for whereas it was estimated that traditional design would have taken approximately five person-months to develop. Perhaps even more impressive was the time it took the team to redesign the antenna after learning that the prototype hardware the antenna was set to be mounted on had changed. Instead of redesigning and testing new antenna prototypes they were

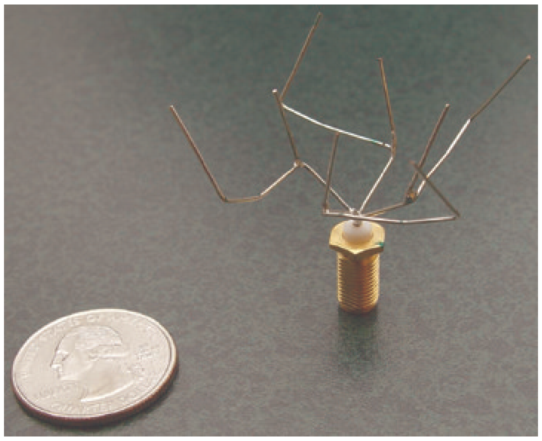

Figure 2: Original evolved antenna

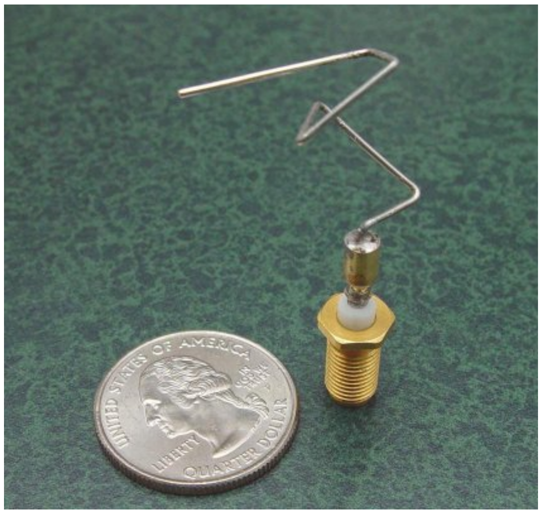

Figure 3: Later generation evolved antenna without branching [5]

able to modify their algorithms and re-evolve a new antenna in only four months [5]. Antenna technology has been covered extensively but it is near impossible to imagine a cover-all equation that will guarantee the best antenna design. A problem like this (manageable complexity and a clear goal) is the perfect type of problem for evolutionary hardware. Not surprisingly, the antennas look foreign and very simple. But that is the general form of an EHW solution: something simple, compact, and not yet thought of [13].

In other cases, the solution had already been thought of. J.R. Kota shows in [12] that, incredibly, at least 15 inventions obtained via evolvable hardware infringe on previously held patents ranging from a high-current load circuit to an electronic thermometer.

7. FUTURE PLANS

7.1 Standardization

There is a push to further categorize and standardize the development of evolvable hardware systems. One can see the distinction between extrinsic and intrinsic hardware as a first step towards categorizing the developing field. It can be seen in [4] that there are three requirements for a material to be considered able to perform evolvable hardware tasks. They are:

1. Material needs to be configurable by applying electricity (or any energy)
2. Material needs to affect an incident signal.
3. Material needs to be able to reset to its original configuration.

Any medium hoping to eclipse FPGAs as the go-to implementation for consumer grade evolvable hardware will likely need to complete all three of the above requirements.

7.2 How limitations are being dealt with

FPGAs are a cheap and stable way to implement evolvable hardware. One reason they may eventually become outdated is if someone finds a way around the inherent problem of scalability of evolvable hardware on a different material. However, that is not to say FPGAs have reached their limit by any means. More likely than not, the complexity scaling issues lie in the evolvable hardware roots than in the medium. Steps are being taken to alter evolvable hardware models and the genetic algorithms to remove the complexity bottle neck. For example, models have been proposed that change the way evolvable hardware passes on its genes. Instead of parents having a single crossover with a fixed number of genes, children of varying genes are sometimes born [13].

7.3 Unstable Environments

Intrinsic evolvable hardware lends itself well to systems that must maintain integrity and stability in a changing and uncertain environment. Systems in areas where it is not possible to replace or repair an instrument are areas being looked into for evolvable hardware. Two examples of hypothetical systems are sensors and space probes. Sensors are often used in places that are not easy to commute to, whether that be in a volcano or an underground oil line, so a system that could handle errors and correct itself would be beneficial. Likewise, space probes can run into unexpected radiation and temperature shifts that may harm or alter electrical components. Intrinsic evolvable hardware could be made that sees abnormalities in test inputs (which are run against already known test outputs) and evolve the system to maintain integrity after simple malfunctions [7].

8. CONCLUSION

In this research, we have gone through the short history of evolvable hardware and shown an implementation on a major medium known as a field programmable gate array. Although EHW is young, it has been able to bridge the gap from novelty to real world solutions on many occasions [11]. There are, however, several hurdles currently being worked on which need to be overcame before EHW can provide solutions to general problems. As it stands, EHW has provided many compact and elegant solutions to application-specific problems. Future efforts, hopefully, will prove EHW as a tool used by researchers and end users to solve complex problems.

9. REFERENCES

[1] *Evolvable Hardware: Evolution in FPGAs*. ACME Seminar.

[2] *An Evolvable Hardware FPGA for Adaptive Hardware*, La Jolla, CA , USA, 2000. The 2000 Congress on Evolutionary Computation.

[3] *Safe Intrinsic Evolution of Virtex Devices*, Dept. of Electronics, University of York, Heslington, York, England, 2000. NASA/DoD Workshop on Evolvable Hardware.

[4] *Evolution in materio: Initial Experiments with Liquid Crystal*. IEEE Computer Society Press, 2004.

[5] *An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission*. U.-M. O'Reilly, May 2004.

[6] *Application-Independent Testing of FPGA Interconnects*, volume 24. Application-Independent Testing of FPGA Interconnects, November 2005.

[7] *On Dependability of FPGA-Based Evolvable Hardware Systems That Utilize Virtual Reconfigurable Circuits*. The Third Conference on Computing Frontiers, 2006.

[8] *The Wider Impact of Moore's Law*. IEEE, September 2006.

[9] *FPGA Circuits for Evolvable Hardware*, volume 2. The Second International Symposium on Electrical and Electronics Engineering, ISEEE, 2008.

[10] *FPGA-based Systems for Evolvable Hardware*, volume 3. International Journal of Electrical and Electronics Engineering, 2009.

[11] *A Bird's Eye View of FPGA-based Evolvable Hardware*, Milano, Italy, 2011. 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2011).

[12] J. R. Koza. Genetic programming is an automated invention machine, http://www.genetic-programming.com/inventionmachine.html, 2003.

[13] J. D. Lohn and G. S. Hornby. Evolvable hardware: Using evoluationary conputation to design and optimize hardware systems. *IEEE Computational Intelligence Magazine*, pages 19–27, February 2006.