Design Space Issues for Intrinsic Evolvable Hardware

James Hereford
Murray State University
Murray, KY 42071
James.Hereford@murraystate.edu

David Gwaltney
NASA-Marshall Space Flight Center
Huntsville, AL 35812
David.A.Gwaltney@nasa.gov

Abstract

This paper discusses the problem of increased programming time for intrinsic evolvable hardware (EHW) as the complexity of the circuit grows. We develop equations for the size of the population, *n*, and the number of generations required for the population to converge, *ngen*, based on L, the length of the programming string. We show that the processing time of the computer becomes negligible for intrinsic EHW since the selection/crossover/mutation steps are only done once per generation, suggesting there is room for use of more complex evolutionary algorithms in intrinsic EHW. Finally, we review the state of the practice and discuss the notion of a system design approach for intrinsic EHW.

1.0 Introduction

This paper looks at the cycle time for the evolutionary process for intrinsic evolvable hardware (EHW), and its relationship to the complexity of the design being sought. In general, intrinsic EHW is successful in designing small- or medium-sized circuits. But the devices themselves, such as Field Programmable Gate Arrays (FPGAs), can implement complex circuits. In the past, researchers have limited the search space when they use reconfigurable devices with large design spaces.

Initially, we define the design space as the total space of allowable circuits given the hardware device(s). In evolvable hardware, the design space is defined by both the number of programmable "configuration blocks" and the number of programmable interconnections among the blocks. To make this discussion general, a configuration block can be as simple as a transistor or as complicated as an amplifier or logic block. The larger the design space then the more complex circuit that can be evolved. We will assume that size of the design space is proportional to the number of programming bits.

The outline for this paper is as follows: Section 2 will develop theoretical equations for the population size and the number of generations required to converge based on the length of the programming string (chromosome). Section 3 discusses the timing of intrinsic EHW experiments. For both a slow and a fast EHW configuration, it is shown that increasing the processor speed alone will not significantly affect the overall timing. Section 4 looks at the current state of the practice of hardware evolution and discusses design approaches that may enable the evolution of complex circuitry. Section 5 gives the main conclusions from the paper.

2.0 Population size and number of generations

To evolve a complicated circuit requires a large design space and hence a long string of programming bits, L. The size of the design space is 2^L, but an evolutionary algorithm (EA) generally does not need to evaluate all of the possibilities to reach the peak or optimum operating point. To determine the number of evaluations that are required for a programming string with L bits, we will consider the size of the population, *n*, and the number of generations, *ngen*, separately. The objective is to determine how *n* and *ngen* scale with the length of the programming string, L, or chromosome length. The total number of evaluations will then be *n*ngen*.

This paper follows the approach of Harik et al. [1] to determine population size, *n*, in terms of length of the chromosome string L for a Genetic Algorithm (GA). Harik develops a model of GAs based on an analogy between genetic algorithms and one-dimensional random walks. The result is an equation that relates the size of the population with the desired quality of the solution, as well as the problem size and difficulty. The final equation for *n* is given by

$$n = 2^{k-1} \ln(\alpha) \frac{\sigma_{bb}\sqrt{\pi m'}}{d} \qquad (1)$$

where

α = probability of GA failure
k = building block order (length)
m' = m-1 = (number of building blocks) – 1
d = signal-to-fitness difference
σ_{bb} = average root mean square building block standard deviation

Equation 1 indicates a GA scales very well to the problem size. It shows that the required population grows with the square root of the size of the problem. For our purposes, we assumed that the programming strings are binary (or binary alphabets), the building block size is 4 (so m = L/4), the probability of GA failure is .01, the fitness difference between the best and 2nd best building block is 1, and the root mean square fitness variance (σ_{bb}^2) is 0.04. (Note: the fitness variance is small compared to the test cases presented by Harik; this value was chosen because it gave pretty good agreement with our evolvable hardware experiments.) This yields an equation for population size as

$$n = 13\sqrt{\frac{L}{4} - 1}\,, \tag{2}$$

where L is the length of the programming string. This result shows that n scales as the square root of L. The constant (13 in this case) will vary based on the problem difficulty (in general it will be a larger number) but n is still $O(L^{1/2})$.

The parameter *ngen* is the number of generations till the population converges. Goldberg [2] utilizes the schema theorem to estimate the number of generations till convergence as

$$ngen = \frac{r+1}{r-1}\ln(n-1)\,, \tag{3}$$

where r is the fitness ratio between a "good" schema and a bad schema. We assume that as the programming string gets longer that the fitness ratio, r, will get smaller. Substituting for r into equation 3 and simplifying yields the number of generations till convergence as

$$ngen = (2^{(\log L+1)} + 1)\ln(n-1). \tag{4}$$

We can now estimate how long it takes to program a device using evolvable hardware techniques. The following chart (table 1) lists several reconfigurable devices that are used for intrinsic evolvable hardware. The table lists the devices, L (the number of bits required to do the programming), 2^L (the total number of possible combinations given the number of programming bits), n, *ngen*, *n*ngen*, and estimated time to program device assuming 0.5 seconds per evaluation. The evaluation time varies considerably in intrinsic EHW based on application and experimental setup; 0.5 seconds is close to the worst case condition.

To determine the computational complexity of the *ngen* calculation, the growth in *ngen* based on L is done via linear regression. A plot of log(L) vs log(*ngen*) is a straight line. The slope of the line, calculated using linear regression, is .43 so *ngen* is $O(L^{.43})$.

Device	**L**	**2^L**	**n**	**ngen**	**n*ngen**	**Time**
Pac30 (amplifier gains only)	20	10^6	26	19	494	4 mins
Pac30 (complete config)	112	$5x10^{33}$	68	39	2652	22 mins
FPTA2 (limited)	500	$3x10^{150}$	145	70	10,150	85 mins
FPTA2 (complete)	5000	$>10^{300}$	459	165	75,735	10.5 hrs
Virtex XCV50 (limited)	9600	$>10^{300}$	637	210	133,770	18.5 hrs
Virtex XCV50 (complete)	1569,000	$>10^{300}$	4900	926	$4.5x10^6$	625 hrs

Table 1: Programming times for reconfigurable devices used for intrinsic EHW assuming 0.5 seconds for evaluation of each solution.

Table 1 makes clear the main benefit of using GAs/evolvable hardware techniques: as the design space grows to nearly infinite proportions ($>10^{300}$ possibilities), the number of evaluations, *n*ngen*, only grows as $O(L^{1/2})*O(L^{.43}) = O(L^{0.93})$ or slightly *less* than O(L).

3.0 Reduction of execution time

In general, an EA used in evolvable hardware has the following major steps: evaluate each member of the population, selection, crossover, and mutation. For intrinsic EHW all of the steps can be done on the processing computer except the evaluation. The evaluation of each member requires several steps: download bit string to device, update device, measure new output, read output and transfer to processor. We define the evaluation time as $t_{evaluate}$.

If the population size is denoted n, then the calculation time for one generation of the EA is

$$t_{cycle} = n*(t_{evaluate}) + t_{processor}, \tag{5}$$

where $t_{processor}$ is the time for the processor to do selection, crossover, and mutation. Consider the impact on the calculation time due to increased processor speeds. The issue is that processor speed/time ($t_{processor}$) is only one part of the equation, so improvements in processor speed only affect one part of the whole process. By analogy with computer design, we can apply Amdahl's Law [3]. Assume that processor speed increases by a factor *nspeedup*. Then Amdahl's Law says that

$$t_{cycle}(\text{after speedup}) = \frac{t_{process}}{nspeedup} + n^* \, t_{evaluate} \quad (6)$$

In other words, t_{cycle} in not reduced by *nspeedup*; rather only $t_{process}$ is reduced by *nspeedup*. Since $t_{process}$ is only a fraction of the t_{cycle}, increasing the processor speed has only minimal affect on the overall time.

To illustrate the (lack of) effect of increasing computer speed, we look at some sample numbers for two different intrinsic EHW setups. One setup utilizes a FPAA from Lattice Semiconductor (ispPAC30) controlled by a PC. A digital multimeter (DMM) is used to read the output and a GPIB (IEEE 488) bus is used to transfer the data to the PC. This is an EHW setup using relatively slow devices.

The FPAA setup has an evaluation time of 35 msec (mainly due to the time to measure the output and transfer the data to the processor). From timing measurements on a Pentium III computer, we know that $t_{process}$ = .6 msec. Calculating t_{cycle} from these values (assuming a population size of 50) yields

$$t_{cycle} = 50*(35) + .6 = 1750.6 \text{ msec} = 1.75 \text{ sec.}$$

From the calculation of t_{cycle}, it is clear that the processor time is a very small fraction of the total cycle time. Even if we used a new and fast computer that was, say 5 times faster (*nspeedup* = 5) than the Pentium III, Amdahl's law says that the new execution time will be

$$t_{cycle}(\text{fast computer}) = 0.6/5 + 1750 = 1750.12 \text{ ms}$$

Therefore, the speedup in processor speed will have an insignificant impact on total time in intrinsic evolvable hardware experiments. Clearly, much more can be gained in this experiment by decreasing the evaluation time.

The second intrinsic EHW setup is the stand-alone board-level evolvable system (SABLES) developed by researchers at Jet Propulsion Laboratory [4]. The SABLES system is an example of a fast EHW setup. From Stoica et al. [4], the stimulus/response time is 1.13 msec per population member, and it takes 6 msec to generate a new population. Thus,

$$t_{cycle} = 50 * (1.13 \text{ msec}) + 6 \text{ msec} = 62.5 \text{ ms.}$$

It is clear that the SABLES setup provides an enormous speed advantage over the (slow) FPAA setup – it is approximately thirty times faster. But again Amdahl's Law shows that speeding up (i.e., reducing) the processor time will have only a minor impact on the overall execution time. Again assuming a speedup of 5 in the processor yields

$$t_{cycle}(\text{fast processor}) = 56.5 + 6/5 = 57.5 \text{ msec}$$

or only about an 8% time reduction for a large improvement in processor speed.

We have shown that improvements in processor speed will not significantly impact the total time in intrinsic EHW experiments. This is in contrast to Genetic Programming and other extrinsic EHW experiments where increases in processor speed have led to development of more complicated circuits [5].

4.0 State of the practice

The previous sections provide a quantitative analysis giving two results for intrinsic hardware evolution. One result shows a GA provides a good approach for searching the solution space as the length of the chromosome increases. The second shows the time for processing a population primarily depends on evaluation time for each individual. These results are qualitatively intuitive based on experience and empirical data, but have not been quantified in published literature for specific devices used in intrinsic hardware evolution. In this section we discuss the implication of these results for practical implementation and give several observations about the current state of practice for intrinsic EHW.

The first observation from published literature is that researchers using intrinsic hardware evolution are primarily concerned with hardware platform development. However researchers concerned with design of the evolutionary process use extrinsic hardware evolution. We define the evolutionary process to include an evolutionary algorithm (EA), chromosome mapping, population sizing/initialization and fitness evaluation. Work involving the use of a physical platform routinely employs standard GAs [6,7,8,9]. Researchers developing more complex approaches to the evolutionary process frequently use simulated hardware components. [5,10,11]. One reason for this may be that simulation lends itself to implementation of complex algorithms more easily than a hardware platform with resource and timing constraints. The evolutionary process and the platform are intimately tied together and impose limits on each other. These two components of hardware evolution are a system to be considered together in the design of an intrinsic EHW platform.

The second observation concerns the basis of our first result (equation 2) in the analysis of simplified statistical processes and genetic algorithms with crossover, but no mutation. Goldberg refers to these equations as facet wise models giving insight into the process of applying GAs to problem solving. These models are verified using simulations of the GA and problem to be solved [1,12]. Goldberg, et. al., consider building blocks in a chromosome to be one bit in the case of the simple ONEMAX problem, concerned with convergence of the chromosome string to all ones. In the case of deceptive traps, the problem includes a building block of 6 bits with a local maximum at 000000 and a global maximum at 111111. In this case, the GA is led astray as the fitness improves as the number of zeros increases, but the best fitness is only achieved if all 6 bits are ones. The goal is maximizing fitness of *m* 6-bit deceptive traps in a chromosome of length, L, where $m = L/6$.

Building blocks in intrinsic EHW have physical relevance, and a chromosome may contain building blocks of varying lengths. The problem of autonomous circuit design is more difficult than the ONEMAX problem or a set of repetitive deceptive traps, making the precise application of such analytical models more difficult. The results obtained here for the number of evaluations, *n*ngen*, are estimates and represent the minimum evaluations needed. Extending these equations to provide population sizing and evaluations bound in a hardware evolution problem will provide tools for implementation of an evolvable hardware platform.

A third observation is many approaches for evolving complex circuitry involve using previously evolved circuits or known circuit configurations [5,13,14]. Time for evolution is reduced by using already evolved components, rather than forcing the evolutionary algorithm to re-evolve basic circuits. Humans routinely employ such a strategy in the design of electronic circuits. While such approaches make sense from a practical standpoint, they are frequently viewed as limiting the design space, especially when "fixing" portions of a reconfigurable device is involved.

5.0 Conclusions

The analytical results show intrinsic EHW employing GAs is a good approach for tackling very large problems since as the design space grows very large (near infinite), the processing time only grows at slightly less than O(L). Adjustments to the models will need to be made to create useful tools for selecting population sizes and estimating the time to convergence for a given application. The analytical results clearly show that the processing time for the EA is a very small fraction of the total cycle time. This implies there is room for use of complex EAs in intrinsic EHW.

The published literature acknowledges that the evolutionary process and the platform are intimately tied together. This supports the notion of a system level view of design space that includes design of the evolutionary process and the physical platform. The use of analytical models for insight into the evolutionary process along with a system level view of intrinsic EHW will help enable successful scaling of designs in practical applications.

Acknowledgements

Dr. Hereford would like to thank the Kentucky NSF EPSCoR Research Enhancement Grant program for support. Mr. Gwaltney is supported by a NASA Marshall Space Flight Center Director's Discretionary Fund task.

References:

[1] G. Harik, E. Cantu-Paz , D. Goldberg, B. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations", Evolutionary Computation, vol. 7, number 3, pp. 231 – 253, 1999.
[2] D. Goldberg, K Deb, J. Clark, "Genetic Algorithms, noise, and the sizing of populations", Complex Systems, vol. 6, pp 333 – 362, 1992.
[3] D. Patterson, J. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann, San Francisco, 1998.
[4] A. Stoica, R. Zebulum, M. I. Ferguson, D. Keymeulen, V. Duong, X. Guo, "Evolving circuits in seconds: Experiments with a stand-alone board-level evolvable system", 2002 NASA/DoD Conf. on Evolvable Hardware, July 2002, pp. 67-74.
[5] J. Koza, M. Keane, M. Streeter, W. Mydlowec, J. Yu, G. Lanza, Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Kluwer, 2003.
[6] M. I. Ferguson, R. Zebulum, D. Keymeulen, and A. Stoica, "An Evolvable Hardware Platform Based on DSP and FPTA", Genetic and Evolutionary Computation Conf. (GECCO-2002), July 2002, pp. 145-152.
[7] J. Lohn, G. Larchev, R. DeMara, "Evolutionary Fault Recovery in a Virtex FPGA Using a Representation that Incorporates Routing", International Parallel and Distributed Processing Symposium (IPDPS'03), April 22 - 26, 2003.
[8] P. Layzell, "Reducing Hardware Evolution's Dependency on FPGAs," Proc of MicroNeuro '99, 7th International Conf. on Microelectronics for Neural, Fuzzy and Bio-inspired Systems., CA. April 1999, pp171-178.
[9] J. Langeheine, K. Meier, J. Schemmel, "Intrinsic Evolution of Quasi DC solutions for Transistor Level Analog Electronic Circuits Using a CMOS FTPA Chip," 2002 NASA/DoD Conf. on Evolvable Hardware, pp. 75-84.
[10] J. Lohn, G. Haith, S. Colombana, D. Stassinopoulos, "Towards Evolving Electronic Circuits for Autonomous Space Applications," Proc of the 2000 IEEE Aerospace Conf., , March 2000, pp 476-486, Vol. 5.
[11] J. Bothelo, B. Leonardo, Vieira, F. Pedro and A. Mesquita, "An Experiment on Nonlinear Synthesis Using Evolutionary Techniques Based only on CMOS Transistors," 2003 NASA/DoD Conf. on Evolvable Hardware, pp. 50-58.
[12] D. Goldberg, The Design of Innovation Lessons From and For Competent Genetic Algorithms, Kluwer Academic Publishers, Boston, MA USA, 2002.
[13] A. Stoica, R. Zebulum, D. Keymeulen, M.I. Ferguson, and X. Guo, "Scalability Issues in Evolutionary Synthesis of Electronic Circuits: Lessons Learned and Challenges Ahead," AAAI Spring Symposium on Computational Synthesis, March 24-26, 2003.
[14] V. Vassilev and J. Miller, "Scalability Problems of Digital Circuit Evolution," 2nd NASA/DOD Workshop on Evolvable Hardware, U.S.A., 2000.