Appears in 1998 International Conference on Evolvable Systems

Evolution of Astable Multivibrators *in Silico*

Lorenz Huelsbergen[1] and Edward Rietman[1] and Robert Slous[2]

[1] Bell Laboratories, Lucent Technologies***
[2] Xilinx Inc.†

Abstract. We use evolutionary search to find automatically electronic circuits that toggle an output line at, or close to, a given target frequency. Reconfigurable hardware in the form of field-programmable gate arrays—as opposed to circuit simulation—computes the fitness of a circuit which guides the evolutionary search. We find empirically that oscillating circuits can be evolved that closely approximate some of the supplied target frequencies. Our evolved oscillators alias a harmonic of the target frequency to satisfy the fitness goal. Frequencies of the evolved oscillators were sensitive to temperature and to the physical piece of silicon in which they operate. We posit that such sensitivities may have negative implications for demanding applications of reconfigurable hardware and positive implications for adaptive computing.

1 Introduction

Complex entities—biological and artificial, for example—are in part governed by systolic processes. In many animals, hearts beat at (perhaps variable) frequencies to distribute fluids. Circuits in machines coordinate information flow in step with a local or global clock. Biological oscillation arose from primitive components (molecules) in an environment (physics) via the process of natural selection. Mechanical oscillation arose in computers, and in many other machines, through human design. We seek to understand if evolution can be harnessed to design pieces of computing machines. Toward this end, we are conducting experiments to "evolve" circuits—oscillators (astable multivibrators) in particular—from primitive logic components. Our goals are twofold: The exploration of the capabilities of *in Silico* evolution and the investigation of whether computational circuits based on oscillators can thus be constructed.

The genetic algorithm (GA) [4] is a form of *evolutionary search*. GAs have been shown to perform well as a general optimization technique across a broad range of domains (see Goldberg [2] for examples). The GA maintains a population of individuals (*bit strings*) over a series of *generations*. The initial population is random. Using an externally supplied *fitness function* (environment), the GA selects promising individuals for the next generation. Some such selected individuals are then paired and, with random substrings interchanged, placed in the next generation.

Evolutionary search—most recently in the form of GA-based *genetic programming* (GP)—has been used to evolve computer software (*e.g.*, [6]). In this context, the bit string comprising an individual is interpreted as a (perhaps variable length) sequence of instructions written in a computer language. Distance, in some metric space, between the result of evaluating a GP individual and the desired target result constitutes an individual's fitness. It is well known that digital software and hardware are computationally equivalent. This suggests that application of software evolution techniques may also be fruitful in a hardware realm [8, 3, 1, 12]. With evolvable computational structures, the programming onus shifts from providing an algorithm (circuit or program) for solving the task at hand to crafting a function that assigns an accurate fitness measure to partial and complete solutions. Search—taking the form of a GA for this paper—can then automatically perform algorithm discovery.

The recent experiments of Thompson [8] in particular demonstrate that reconfigurable logic in the form of field-programmable gate arrays (FPGAs) can serve as a viable substrate for gate-level hardware evolution. Thompson evolved discriminator circuits that, when presented with one of two possible input tones (frequencies), would correctly classify their input. Our system for *in Silico* evolution is similar to Thompson's. Our

*** {lorenz,ear}@bell-labs.com
† robert.slous@xilinx.com

study however concerns circuits with fundamentally different characteristics than input-sensitive frequency discriminators—we are evolving computational components, namely oscillators, that function as stand-alone clocks.

Our result is the automatic generation of oscillators at specified *target frequencies* from primitive electronic components. Given only a target frequency f, our system can produce—from logic gates (such as "`not`") and wires to connect them—a circuit whose single output oscillates between the logic states low and high with frequency f (or a harmonic multiple thereof). Note that the circuit undergoing evolution receives no input in general and no clock signal in particular. It completely synthesizes oscillation.

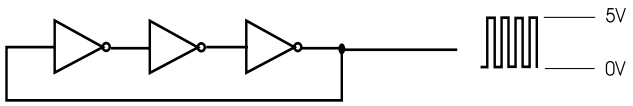


Fig. 1.: A manually designed ring oscillator constructed from three inverters. The output oscillates due to gate and signal propagation delays. A genetic algorithm can find circuits with similar behavior—but not necessarily of similar structure—that oscillate at predefined target frequencies.

One route to such oscillation is exploitation of gate and signal propagation delays along with feedback. The manually designed ring oscillator of Figure 1 illustrates this basic principle. This oscillator's frequency depends on the speed of the substrate's implementation technology.[5] Note that the oscillator's frequency may be reduced by inserting additional inverters into the ring. It is difficult, even for skilled designers, to craft feedback circuits with specified characteristics (*e.g.*, frequency) solely from logic gates. In practice, therefore, oscillators are usually constructed from (relatively expensive) analog components.

Though most often deployed in digital circuits, transistors are analog amplifiers. Since gates are constructed from transistors, circuit evolution is free to exploit their analog behavior to satisfy its goal. Furthermore, since electronic components are physical devices that operate electromagnetically, effects such as cross coupling are also available to evolution. As we will discuss (§5), and Thompson considers at length [10, 9], exploitation of such "features" poses new engineering concerns that must be addressed. On the other hand, we observe functional portability of evolved circuits from one piece of silicon to another which discounts the wide-spread exploitation of such chip-specific peculiarities.

We report results of *in Silico* oscillator evolution for ten target frequencies in three cell-array sizes (6x8, 8x8, and 16x16). Considering all three cell-array sizes, our system discovered quite accurate oscillators—over 97% of their pulses correct—for five of the ten frequencies and required only a small number of GA runs. Our empirical results are statistically reproducible. Random search confirms that the directed search of the GA in the space of oscillator circuits is effective; that is, the evolutionary search is directed and does not "blindly stumble" upon solutions.

2 Related Work

Although all experiments on hardware evolution are recent, the idea [12, 1] spans at least two decades. To date a few groups have used software simulation (digital and analog) to provide circuit fitness measures [3, 5]. Aside from the work of this paper, one other experiment [8] used reconfigurable logic to directly, in real time, to evolve circuits *in Silico*. We now provide further details on the prior work.

Wolfram [12] and Atmar [1] are among the first to describe frameworks based on (evolutionary) search to create computational hardware. Wolfram posits that it may be possible to find a realization of simple combinational circuits using genetic search and, thereby, to mitigate the complexities of hardware design. In his thesis, Atmar considers hardware that admits rapid evolution of finite state machines. Since the design and construction of such hardware was relatively expensive at that time, Atmar simulated his approach in software. Though slow, the simulation was applied with some success to the task of character recognition.

[5] Gates in contemporaneous off-the-shelf FPGAs typically switch in nanoseconds.

The most significant experimental result to date is that of Thompson [8]. He used a reconfigurable hardware device (FPGA) to evolve a binary frequency discriminator that classifies its input signal into one of two frequencies. His search was also GA based and obtained, as we do here, its fitness measures in real time from the FPGA. Thompson identified the phenomenon of evolution exploiting the device's analog nature (*i.e.*, the physics of its immediate environment) in finding solution circuits [9] (see also Section 5 below). Earlier, Thompson *et al.* [11] simulated oscillator evolution in a software model of an FPGA. They report evolving pulse trains with frequencies much lower than speeds of the simulated logic gates, but not at the target frequency. Because of simulation overheads, circuit evaluation times were limited to 10ms; our experience reveals that times an order of magnitude longer are necessary to yield stable oscillators in practice. Our work differs from Thompson *et al.*'s [11] in three respects: (1) our evolved oscillators function in an actual reconfigurable device; (2) their frequencies are very close to a multiple of the target frequency; and, (3) they are stable over time (*i.e.*, for many hours).

Others have coupled GAs to electronic and electrical circuit simulators as sources of fitness measures. Koza *et al.* (*e.g.*, [5]) coopted genetic programming of software to evolve instead descriptions of analog electrical circuits. They however do not report on the conversion of evolved descriptions to actual hardware. It has not been demonstrated that analog circuits evolved via simulation are implementable in practice. *In Silico* evolution, on the other hand, assures that solution circuits function in at least the device (and environment) in which they originally evolved. Higuchi *et al.* (*e.g.*, [3]) similarly use simulation of, in their case, digital circuits to guide evolution. They do test and verify their evolved circuits in an FPGA, but only after evolution is complete. *In Silico* evolution—Thompson's [8] and our approach—can speed circuit evolution since evaluation is in hardware proper. The *in Silico* approach also provides evolution the ability to exploit analog circuit characteristics whereas digital simulation abstracts this physics. Higuchi *et al.* [3] describe various means by which hardware evolution can be extended; by performing the genetic search in hardware instead of software, for example.

3 System for *in Silico* Evolution

Our custom system for *in Silico* gate-level evolution consists of a hardware component coupled with a GA implementation. We first describe the hardware and then the software; further details of the hardware are available elsewhere [7].

3.1 Hardware: FPGA and PC

The hardware consists of two pieces: an FPGA used exclusively for assigning fitness values to circuits and a general purpose computer (PC) that services the FPGA and executes the search algorithm. Figure 2 is a block diagram of this system.

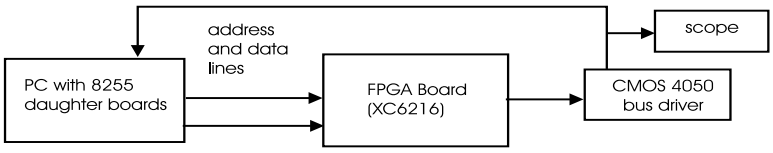


Fig. 2.: A system for *in Silico* evolution. The PC downloads configurations (circuits) into the FPGA for fitness evaluation. It also samples and records the FPGA's output. The bus driver insures that the FPGA output is forced either completely low or high. Peripheral IO (PIO) devices consitute the PC/FPGA interface.

An FPGA is an electronic device (usually packaged as a single chip) that can be configured to function as an arbitrary digital circuit.[6] Reconfigurability can be achieved by associating memory cells with logic gates and interconnections. The states of the configuration cells govern a gate's specific logic function or an interconnection's source and destinations. That is, a circuit for embodiment in an FPGA can be described

[6] The FPGA's size limits the digital circuits it may realize.

by a bit string that contains information on the type of gates a circuit requires and how they are to be connected. The genetic algorithm described in the next section will search for bit strings describing circuits that induce a target behavior (*i.e.*, oscillation) in the FPGA.

FPGAs generally consist of a collection of *cells* along with one or more levels of hierarchical interconnect among the cells. Additionally, FPGAs contain IO blocks (IOBs) that enable the configured circuit to communicate with external devices.

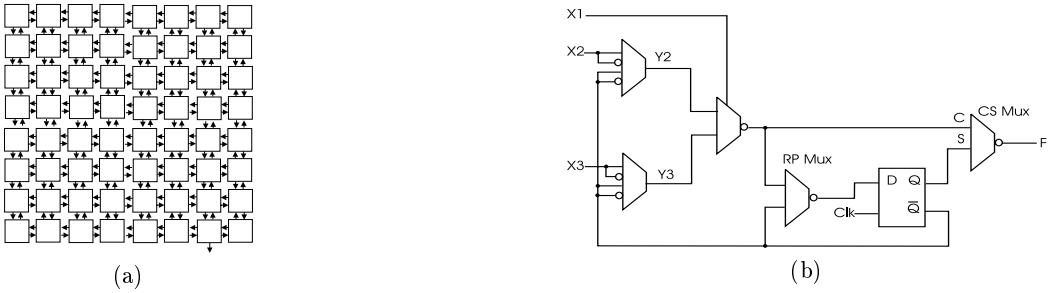


Fig. 3.: Relevant portions of the XC6216 FPGA architecture. Sample logic-cell layout (a) used for the experiments of this paper. Nearest-neighbor connections and the single bit-wide output line (lower right) are depicted. Digital per-cell logic (b) consists of multiplexors controlled via configuration bits.

Figure 3 contains high and low level views of the Xilinx XC6216 architecture relevant to our experiments. Figure 3a depicts a sub-array of the FPGA's cells and enabled connections (nearest-neighbor). Note the bit-wide output line emanating from a cell on the bottom row. The oscillator experiments of this paper were conducted in such 6x8, 8x8, and 16x16 arrays with single outputs. The output of a single cell was routed to a XC6216 IOB which transmitted the signal to the PC as well as to an oscilloscope for monitoring. Note that—among other points discussed below—we utilize only the lowest level of inter-cell communication and that the FPGA circuit receives no input (but may produce a time varying output signal).

The Xilinx XC6216 FPGA [13] was chosen for our implementation for a number of reasons:

1. it may be reconfigured indefinitely,
2. it may be partially reconfigured,
3. its design insures that an invalid circuit will not harm it,
4. its specification is non-proprietary.

Point 1 allows its use in repeated fitness evaluation which is the central component of a GA.[7] Point 2 guarantees that configuration time (a non-negligible cost in fitness computation) is linear in the size of the circuit being loaded. Point 3 tolerates random circuit configurations which are prevalent in the circuit population maintained by the GA. Point 4 permits bit-level circuit description while bypassing conventional circuit entry tools.

Figure 3b is a schematic of the digital logic within a single XC6216 cell. Multiplexors, controlled via configuration bits, implement common logic functions. Wires into a cell (from its nearest neighbors) constitute its $X1$, $X2$, and $X3$ inputs which, by appropriate configuration of the multiplexors, can compute all $2:1$ and some $3:1$ boolean functions for the cell's output, F. A one-bit register is also available per cell. However, we disabled all registers for this paper's experiments. (Evolution is free to construct register structures by appropriately connecting multiple cells.) Configuration of a single XC6216 cell nominally requires three bytes of information.

It is important to note that although the cell function is described in terms of digital logic, the cell is constructed from gates which are in turn built from analog components (transistors). Thus, a circuit constructed in an unconventional manner may exhibit analog behavior that may extend to the global (circuit)

[7] Our chip has been reconfigured millions of times.

level. GA fitness evaluation is an unconventional FPGA application and we make no attempt to detect or restrict configuration strings describing such circuits.

3.2 Software: Search Algorithms

This section describes the implementation of the genetic algorithm used in the oscillator experiments and also recounts the random-search algorithm used to verify the efficacy of *in Silico* evolution. We defer elaboration of the various constants (*e.g.*, population and sample sizes) to Section 4.

Evaluation Function. Fitness assignment requires an assessment of an individual's performance on some particular task. This is done via an *evaluation function*:

$$\mathcal{E}_O : (I, f) \rightarrow S \tag{1}$$

$\mathcal{E}_O$ maps an n-bit configuration string (individual) I and a frequency f to an m element *sample vector* S. This vector contains m output values (each either low or high) of the FPGA's output line at $1/f$ second intervals and is used in assigning individual I's fitness.

We note that in our implementation the frequency f may only be instantiated to values that can be readily synthesized in software by the PC (see §4).

Fitness Function. A circuit I's fitness is computed using only the sample vector S returned by the evaluation function (Equation 1). Lower fitness values are better. The fitness is given by:

$$\mathcal{F}_o\,(S\,) \equiv \sum_{i=1}^{m} S_i \oplus (i \bmod 2) \tag{2}$$

where S_i is the i-th element of the sample and $\oplus$ denotes "exclusive or." The fitness, therefore, is the number of "missed" pulses. A missed pulse occurs when an even (odd) sample element does not match a low (high) target pulse.

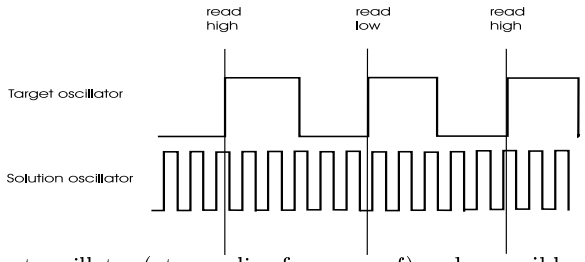


Fig. 4.: Juxtapostion of the target oscillator (at sampling frequency f) and a possible solution oscillator at a harmonic of f.

Figure 4 illustrates the fitness computation process. The top pulse (of sample frequency f) indicates the points in time when samples are taken. The FPGA's output is always sampled at the beginning of a high pulse and alternating samples are taken to be high followed by low, *etc.* Note that an oscillator of a harmonic multiple of f may satisfy the desired criteria.

Search Methods. Here, we describe details of the genetic and random search.

Genetic Search. We use *tournament selection* with *elitism* as the GA's population-selection mechanism. *Crossover* and *mutation* are its genetic operators.

Population selection, for the construction of successive generations, is performed via k-tournament selection. Let P be a population (set) of N individuals (configuration strings). To select a single individual from

P, tournament selection examines k individuals in P and selects the one with best fitness. This mechanism is used to generate candidate pairs for crossover, for example.

We use a recombination operator that performs two-point crossover. Crossover of two n-bit configuration strings I_a and I_b first selects a subsequence of bits[8] starting at a random point $0 \leq p_a < n$ in configuration I_a. The length $k > 0$ of the subsequence is chosen randomly such that $p_a + k < n$. A random point p_b, $0 \leq p_b + k < n$, is then chosen in configuration I_b. Finally, the k bits in I_a starting at p_a are interchanged with the k bits in I_b starting at p_b.

Crossover in a population P is performed by first selecting a subset $P' \subseteq P$ of configurations from the population (using tournament selection). A configuration is randomly selected for P' with probability $Prob_{crossover}$. The configurations in P' are then randomly paired and the crossover operator is applied to each pair to produce a replacement pair.

Bits in every configuration string in P are mutated (negated) with probability $Prob_{mutate}$ during generation transitions.

Random Search. Random search randomly generates an individual I, evaluates I and computes its fitness, and (optionally) records I's fitness as the best seen if I improves on the current best fitness. This process continues until a sufficient number of global solutions are found or until the number of configuration evaluations exceeds a predetermined threshold.

4 Experiments

For three array sizes—6x8, 8x8, and 16x16—we performed oscillator evolution experiments for ten target frequencies in a Xilinx XC6216 FPGA [13]. Each of the thirty experiments consisted of ten randomly seeded GA runs. Table 1 lists the frequencies and summarizes the results. Note that the evolved oscillators are close to harmonics of the target frequency. The highest target frequency is the one at which the PC can most rapidly sample the FPGA by polling the IO cards mapped to the PC's port. The nine lower frequency targets were obtained by successively inserting blocks of 100 null operations (NOPs) into this polling loop. (See §5 below for alternatives to software sampling.) Actual frequency determination was then done with an external oscilloscope attached to the bus driver output.

f	f'	GA	Random
88.4		4949	–
66.3		8298	–
55.3		5527	–
47.4		5475	–
40.1	925	149	9943
35.2		6808	–
31.5		8808	–
28.4		6475	–
25.9		6947	–
23.7	925	220	9475

6x8 Cell Array

f	f'	GA	Random
88.4		4381	–
66.3	1185	466	9734
55.3	929	213	9099
47.4		6032	–
40.1		5025	–
35.2		7066	–
31.5		8772	–
28.4	909	341	9905
25.9		6535	–
23.7		8016	–

8x8 Cell Array

f	f'	GA	Random
88.4		6033	–
66.3	909	575	9931
55.3	925	52	8313
47.4		6920	–
40.1		4134	–
35.2		8194	–
31.5		7523	–
28.4	943	552	9886
25.9		6491	–
23.7		8533	–

16x16 Cell Array

Table 1.: Results of evolving oscillators for ten target frequencies in three cell-array sizes. The target frequency f and the evolved frequecy f' are in kHz. (f' is given only for experiments that produced fitness values less than 1000.) The table lists the best fitness (as the number of missed pulses out of 2×10^4) found by the GA and, for targets that gave oscillators, by a random search.

GA parameters were set as follows: population size $N = 512$, $Prob_{crossover} = 25\%$, $Prob_{mutate} = 0.01\%$. The probability settings mean that a quarter of every generation was generated through crossover (the

[8] To simplify implementation, we select points at byte boundaries. We do not expect that this simplification fundamentally affects our results.

other three quarters being directly selected) and that one bit (randomly selected) in every 10^4 bits in the population was negated during transition into a new generation. Binary tournament selection was used. A GA run was deemed complete when the fitness of the population's best individual did not improve for 50 consecutive generations (stasis). We have no reason to believe these settings to be in any way "optimal" for the problem under investigation; alternate settings have not been tried.

The length of the configuration string was 576 bytes of which 1920 bits (41.6%) actively controlled the FPGA. To simplify the problem, the remaining bits were masked to enable only nearest-neighbor interconnections and to disable cell registers. We chose to disable these features to simplify the problem and hence do not yet have data on whether their inclusion affects the solutions.

In a fitness evaluation, the FPGA's output line was sampled $m = 2 \times 10^4$ times at the target rate. We arrived at this sample size by noting that shorter samples (*e.g*, $m = 1 \times 10^3$) produced unstable oscillators. That is, oscillators evolved using a significantly shorter sample sequence decayed after producing pulses only for the duration of the sampling. We find that the GA can encourage oscillator stability by integrating over longer sample times. In light of these findings, it would be interesting to examine the stability of Thompson *et al.*'s [11] oscillators evolved under software simulation.

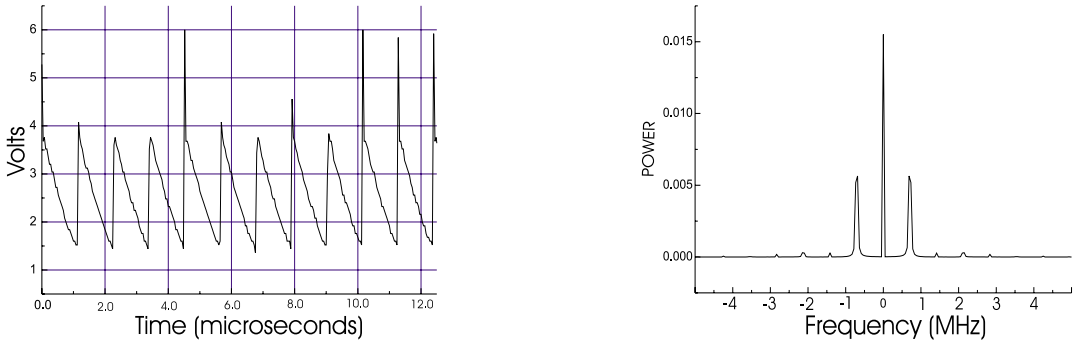
For each target frequency f, Table 1 records the best GA individual fitness (fewest missed pulses) found for the three array sizes over ten runs. For individuals with fitness less than 1000 we list the oscillating frequency f'. The fitness of the best individual found via random search is also given. Our results are statistically reproducible. This means that in a fixed number of runs, evolutionary searches will find circuits with comparable fitness even though the searches' random seeds differ.

The GA found oscillators of good, but not perfect, accuracy. Solutions always appeared within the first 300 generations. Note that all evolved oscillators are close to a harmonic of the target frequency. The oscillators found in the smallest array (6x8) are qualitatively different than those found in the larger (8x8 and 16x16) arrays in that they were found at different frequencies. In particular, the evolved 6x8 oscillator for target 40.1kHz has a frequency similar to the 16x16 evolved oscillator for the 55.3kHz target; yet no oscillator was found in the 6x8 cell space for the latter target. That is, the 6x8 oscillator for 40.1kHz should also be an oscillator for 55.3kHz in the same cell space; yet it was not found for target 55.3kHz. Restriction of resources apparently enables discovery of certain oscillators but also inhibits discovery of others. A possible explanation is that the restricted cell space blocks some evolutionary pathways that are available with more resources.

The evolved oscillator circuits require further study to discern their internal structure. It is straightforward to construct—from the configuration bit strings—a schematic of the cells' functions and the wires that connect them. However, it is unlikely that a digital implementation (with different layout, for example) constructed from a disassembled schematic will yield identical, or even similar, behavior. In other words, it is possible that the evolved circuits exploit parallel interactions between gates used in an asynchronous uncontrolled fashion. In the oscillator experiments we observe that the the frequency of the oscillator circuits could be governed by such effects but not their general function (oscillation); we conclude this because the circuits portably oscillate in different silicon pieces, but at different frequencies (see §5).

Figure 5a depicts an oscilloscope output of the 929kHz 8x8 oscillator found during the search at target frequency $f = 55.3$kHz. The voltage on the FPGA output line was captured *before* the signal passed through the bus driver (Figure 2). Capture of the output before the driver yields a sawtooth function that oscillates between TTL threshold voltage levels. Note that the circuit's gates are exploiting threshold levels to achieve oscillation. The signal *after* the bus driver approaches the form of a square wave (not shown). Figure 5b is the power spectrum for this sawtooth signal; it indicates that the signal lies mainly about 900kHz, but also contains a few high-frequency components. In other words, the oscillator is not spectrally pure, but quite close.

To assure that evolutionary search was indeed approaching solutions and not randomly "stumbling" upon them, we conducted 10^6 random-search evaluations (§3.2) for each experiment that yielded a circuit with GA fitness below 1000. The best fitness produced by random search never reached 9000. (In comparison, ten GA runs for a given frequency required approximately 3.5×10^5 evaluations.) Note that a circuit that holds its output either low or high will trivially score a fitness of $m/2$ (where m is the number of pulse samples). Random search—even when allotted more evaluations than the GA—has not located a circuit whose output was in the correct state more than 52% of the time. Furthermore, evolutionary search always found circuits closer to the goal while requiring fewer evaluations.



(a) (b)

Fig. 5.: Evolved oscillator and its power spectrum. Oscillator (a) of frequency 929 kHz, a harmonic of the target frequency $f = 55.3$ kHz; its pulses coincide with 98% of f's pulses. The power spectrum (b) indicates that the signal's dominant components lie at the measured harmonic frequency.

5 Discussion

We conclude this paper with a discussion of the results and of some engineering issues raised by *in Silico* evolution.

Evolutionary search discovered oscillators for some, but not all, presented target frequencies. It is desirable, however, to develop methods that can evolve oscillation for more—if not for most—frequencies. We believe oscillator circuits were not found for some frequencies for two reasons: (1) oscillation at the target frequency is not possible given the available resources and (2) the length of the search was insufficient. Comparison of array sizes 6x8, 8x8, and 16x16 was done to begin investigation of the first point. Since the size of the search space grows exponentially in the size of the hardware array, searches in larger arrays will likely be more computationally expensive. (For the 8x8 array the search space already consists of 2^{1920} possible configuration strings, some of which describe semantically equivalent circuits; this size increases by a factor of 16 for the 16x16 array.) Conducting similar experiments with larger populations or greater stasis settings would expand the search which—in exchange for longer run times—could turn up other oscillating circuits

Our method of sampling the FPGA with a software polling loop running on a PC introduced some indeterminacy—such as operating system interrupts—into the fitness measurement process. Future systems for oscillator evolution could employ dedicated frequency generators and microcontrollers to better control the sample intervals and to provide a larger and finer spectrum of target frequencies. Software sampling did not however pose a significant obstacle to evolving some fairly accurate oscillators; better sampling technique may yield more accurate oscillators or even oscillators for frequencies so far found untenable by evolutionary search.

It is not at all understood *how* the evolved circuits function. For example, relative to the speed of the FPGA's gates (nanosecond transition times), the evolved oscillators are of rather low frequency. To explore this point, we manually constructed the largest (63 inverters), and hence slowest, ring oscillator that fits in the 8x8 cell space. (Figure 1 depicts a similar three-inverter ring.) This oscillator has a frequency of 3560kHz which is still much higher than the frequencies of the evolved oscillators in all three array sizes. Our evolved circuits must therefore be using mechanisms other than simple ring oscillation to attain their behavior. Disassembly of the circuits into digital schematics may provide some clues, but it is unlikely that this will be sufficient to fully understand their behavior.

Evolution operates with respect to an environment. *In Silico* circuit evolution occurs not only in the silicon of the reconfigurable logic device, but also in the environment where that device is located. Like Thompson [10], we have observed that temperature affects the evolved circuits. Cooling the FPGA increases the frequency of oscillation in our case. The oscillators of this paper were evolved at "room temperature." Good temperature control seems paramount to evolving more precise circuits. Thompson [10] is investigating solutions using simultaneous evolution in multiple environments to increase circuit applicability and robustness. Other factors, such as the position of the circuit within the FPGA's full 64x64 array, or the silicon wafer from which the particular FPGA was cut, may have an effect on the portability of evolved circuits.

We verified that it is possible to use an oscillator evolved in FPGA chip A in another chip B (of the same variety)—albeit with a slight change in frequency.

The influence of factors such as temperature and silicon quality on circuit evolution is an impediment to the rigorous adherence to specification (of frequency for example) required by conventional system design. However, evolved circuits are potentially quite adaptive. For example, the evolved oscillators function as thermometers that could control the device over a temperature range. Evolution is an adaptive process: it could potentially salvage defective chips by evolving circuitry to avoid—or even harness—the defects.

6 Summary

We have described a system for using evolutionary search to find electronic circuits that approximate or meet a given specification—in particular, to find oscillators at various target frequencies. Real-time fitness measures, obtained from a reconfigurable FPGA, guide the genetic algorithm that performs the search. For five of ten target frequencies, the system was able to construct circuits that oscillated at a harmonic close to the target frequency. Automated search algorithms operating in the space of circuits can be harnessed as a powerful new aid in the design and construction of certain electronic hardware.

Acknowledgments

Thanks to Bob Frye for valuable insights into this work. The anonymous referees, Brian Kernighan, and Adrian Thompson supplied useful comments.

References

1. J. W. Atmar. *Speculation on the Evolution of Intelligence and its Possible Realization in Machine Form.* PhD thesis, New Mexico State University, April 1976.
2. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.
3. T. Higuchi, H. Iba, and B. Manderick. Evolvable hardware with genetic learning. In H. Kitano and J. A. Hendler, editors, *Massively Parallel Artificial Intelligence*, pages 399–421. MIT Press, 1994.
4. J. Holland. *Adapation in Natural and Artifical Systems.* University of Michigan Press, 1975.
5. J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Automated WYWIWYG design of both the topology and component values of analog electrical circuits using genetic programming. In *Proceedings of the First Conference on Genetic Programming*, pages 123–131. MIT Press, July 1996.
6. J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors. *Proceedings of the Second Genetic Programming Conference.* Morgan Kaufmann, July 1997.
7. E. Rietman, R. Slous, H. Hemmi, H. de Garis, and K. Shimohara. Building a machine for evolution *in silico.* In M. Sugisaka, editor, *Proceedings of the Third International Symposium on Artificial Life and Robotics*, pages 186–189, January 1998.
8. A. Thompson. Silicon evolution. In J. Koza, editor, *Proceedings of the First Conference on Genetic Programming*, pages 444–452. MIT Press, July 1996.
9. A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In T. Higuchi and M. Iwata, editors, *First Int. Conference on Evolvable Systems: from Biology to Hardware (ICES96)*, pages 390–405. Springer Verlag LNCS 1259, 1997.
10. A. Thompson. Temperature in natural and artificial systems. In P. Husbands and I. Harvey, editors, *Fourth International Conference on Artificial Life*, pages 388–397. MIT Press, 1997.
11. A. Thompson, I. Harvey, and P. Husbands. Unconstrained evolution and hard consequences. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware: The Evolutionary Engineering Approach*, pages 136–165. Springer–Verlag, 1996.
12. S. Wolfram. Approaches to complexity engineering. *Physica D*, 22:385–399, 1986.
13. Xilinx Inc. *The Programmable Logic Data Book. XC6200 Advanced product specification V1.0*, 1996. `http://www.xilinx.com.`