# Evolving Circuits on a Field Programmable Analog Array Using Genetic Programming

by

Michael A Terry

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2005 [June 2005]

© Massachusetts Institute of Technology 2005. All rights reserved.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 18 2005
LIBRARIES

Author .....
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by.....
.....
Una-May O'Reilly
Principal Research Scientist
Thesis Supervisor

Accepted by .....
.....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

BARKER

# Evolving Circuits on a Field Programmable Analog Array Using Genetic Programming

by

Michael A Terry

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

This thesis describes the design and implementation of the Genetic Programming Intrinsic Circuit (GPIC) design system. Inspired by a number of recent advances in the field of Evolvable Hardware, the intended purpose of GPIC is to automate the design of analog circuits with minimal domain knowledge, computational resources, and cost using Genetic Programming with candidate solutions implemented in real hardware. This system has been constructed out of commercially available hardware and software, and the components were integrated through the development of a modular device-independent software system. The fitness evaluations of the candidate solutions of the Genetic Programming module are realized through a C interface to a National Instruments Data Acquisition Card. This Genetic Programming approach to analog circuit design decreases the fitness evaluation time of previous approaches by substituting expensive circuit simulation for real-time hardware testing. Since performing fitness evaluations in simulation is limited by the known model for a given environment, intrinsic testing provides additional benefit through the inherent incorporation of any unknown environmental conditions during tests. This feature is especially important for autonomous systems in unknown environments, and systems that must perform well in extreme environments.

Thesis Supervisor: Una-May O'Reilly
Title: Principal Research Scientist

# Acknowledgments

The early phases of this project owe many thanks to the guidance and suggestions of many of the folks at the Institute for Defense Analyses: **Brian Cohen**, **Lou Lome**, **Robert Rolfe**, and my 'unofficial' mentor **Daniel Spar**. With their help I was able to turn ideas from a challenging internship into a very enjoyable thesis.

For their contributions to many of the driving ideas of this project, I would also like to acknowledge and thank **Eduardo Torres-Jara**, **Dmitry Berenson**, **Adrian Stoica**, **Didier Keymeulen**, **Garrison Greenwood**, **David Hunter**, and **Anadigm(The FPAA Company)**.

I'd also like to thank my **brother Rudy** for his encouragement and inspiration to work hard, and my **mom** for her general support, and for keeping my expectations grounded in reality.

Finally, I would like to extend my gratitude to **Dr. Una-May O'Reilly** for her endless patience, commitment, and words of wisdom through the ups and downs of this project. I couldn't imagine having a more understanding and thesis advisor.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivations

Inspired by a number of recent developments in the field of Evolvable Hardware, this project commenced in the Fall of 2004 as an investigation into the capabilities and merit of Intrinsic Evolvable Hardware as a routine design methodology. We feel that the combination of evolutionary algorithms and reconfigurable hardware has great potential for achieving very robust systems design, and have thus set forth on the design of a testbench for investigating these capabilities.

We also realized early an interest early in the project in focusing on evolution of analog circuits and applications. Since real signals and systems are analog and continuous in nature, we did not want to restrict ourselves to working on problems abstracted to the digital domain. Also, since evolvable hardware techniques are challenging many traditional methods for routine hardware design, it was our intention to start our investigation at a very fundamental level of abstraction.

Another major goal for this project was to learn the known capabilities and limitations of digital and analog evolvable hardware, to allow us to formulate the right research questions. Since this work represents the first initiatives at the MIT Computer Science and Artificial Intelligence Laboratory in Evolvable Hardware, an extensive literature search was needed to gauge the directions and trends in the field. Understanding that much of the important information could not be extracted solely

from text, it was important to work with existing researchers on overcoming some of the technical challenges of developing this testbench, and to share our insights into our challenges.

Once the next design phase of the testbench are complete, it is my hope that this work will lead to continued initiatives in the area of adaptive control systems through the use of evolvable hardware. As described in Chapter 5, controls is a good match for solutions from Evolutionary Algorithms, primarily since it is a problem of multi-objective optimization.

## 1.2 Requirements

There were a number of bare-minimum requirements we set forth early in the project that served as guiding principles:

- **Human design level abstraction and techniques should not be ignored.** Although an evolutionary approach to design often requires little to no domain knowledge, it is our belief that what is currently known about a given problem provides important clues to reducing the search space.

- **An evolutionary algorithm must be provided with the right building blocks.** Similar to the first guiding principle, we placed a great deal of emphasis on understanding the tradeoffs between choosing a specific set of primitives, or building blocks, for evolutionary search. The goal is to find solutions with minimal computation resources within the search space implied by composition of these primitives, and to reuse known configurations.

- **Parameter or bitstream search is not enough.** Determining the topology of a circuit as well as parametric aspects of a given topology is necessary in order to obtain powerful, scalable solutions to difficult circuit design problems.

- **Physical instantiation of candidate circuits is key.** Although simulation of candidate solutions can be more reliable across reconfigurable devices, the

true value of evolvable hardware comes in evaluating fitness in a real environment. Copmared to simulation, intrinsic instantiation also offers a speedup and reduction of computation resources.

- **Careful consideration should be made to scoping the project correctly.** Although the MIT Computer Science and Artificial Intelligence Laboratory would be able to provide us with a wealth of resources, it is important for the scope of this Master's thesis remain manageable for one graduate student and advisor, less than $5K of funding, and one year's timeframe.

- **Any work should be as extensible and as portable as possible, but not one bit more.** Since this project represents potentially the first of many student projects, we realized the need to keep our hardware and software development, and tools modular and platform-independent. Even though choosing a specific field programmable device dictates the lowest level circuit building blocks, few explicit device-dependent accommodations should be made in the evolutionary algorithm. Future projects may leverage some of our work on any number of platforms, from embedded systems to robots with a variety of field programmable devices implementing solutions. As described in Chapter 3.1, developing a system that incorporates commercial parts while reducing the number of dependencies on specific software packages is a particular challenge. However, it may be necessary to adopt specific dependencies if the alternatives are too costly to development time or other resources.

# Chapter 2

# Background

## 2.1 Evolutionary Computation

What better place to find inspiration for design of robust systems than nature? In observing living systems in our environment, it is clear the great accomplishments natural evolution has achieved. Using the inspiration of neo-Darwinian evolution, Evolutionary Computation is a field of research that aims to solve complex search, design, and/or optimization problems through adaptive, population and genetics-based search. Most of these techniques fall under the umbrella of Evolutionary Algorithms (EA). Early evolutionary computation work in the 1960's [3] established that evolutionary algorithms could be applied to solve problems in a variety of domains, including design automation, robot learning, and the benchmark Traveling Salesman Problem. The most widely known class of EAs fall under the category of Genetic Algorithms (GA). In general, Evolutionary Algorithms, simulate evolution through the natural selection of individuals within a population of candidate solutions, whose characteristics are described by simulated chromosomes. The genes of individuals in each population are mutated and crossed-over with other members to create new solutions, in an iterative procedure [8]. Members of each population "survive" to the next simulated generation if they are valued highly by a specified "fitness" function. After a large number, often millions, of generations, this process can converge to very unique and fascinating designs. Many of these solutions cleverly exploit re-

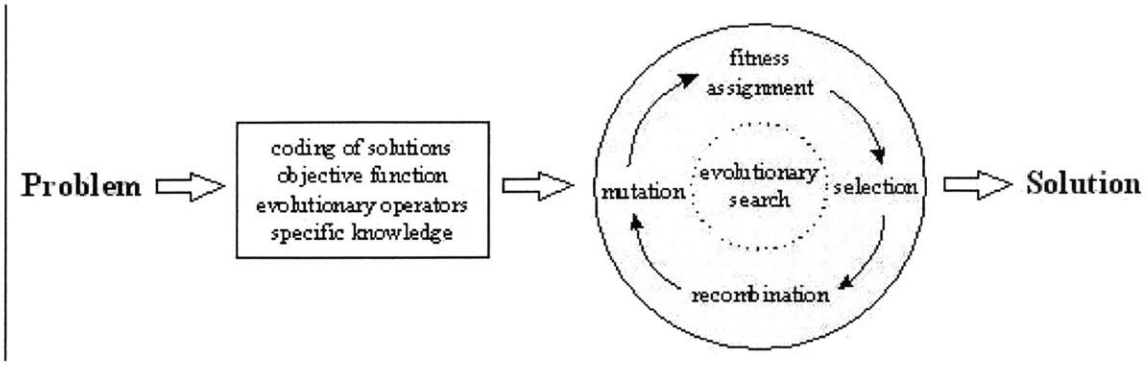


Figure 2-1: Steps in an Evolutionary Algorithm.

sources in the environment, occasionally "cheating" to achieve the solution. The use of computers speeds up the evolutionary process from millions of years to milliseconds. A complete description of evolutionary algorithms, including Genetic Programming, Evolutionary Strategies, and Evolutionary Programming is beyond the scope of this paper. Instead, I will focus on a sub-domain of Evolutionary Computation known as Evolvable Hardware to attack problems in robust system design.

## 2.2 Evolvable Hardware

Evolvable Hardware (EHW) is used to describe the use of Evolutionary Algorithms (EA) to search for solutions for some of the most difficult problems in the design space of electronic circuits and systems. The power of this approach lies in giving the designer the ability to let artificial evolution search for the best solutions to a given specification, without requiring much knowledge of the problem domain. (See [10] for examples of evolved designs that are considered "human competitive" given enough search time and computational power) The main tasks for the designer are: 1) defining an appropriate measure of "fitness" from an abstracted functional level, 2) providing an appropriate functional mapping of chromosomes for the algorithms, and 3) selecting appropriate parameters of the Evolutionary Algorithm for a given application. Some knowledge of the problem domain is critical to completing these tasks successfully.

Another major selling point for evolvable hardware is its unique ability to incorporate any number of environmental factors and process variations when the evolutionary process is carried out in a real environment. This ability becomes incredibly useful when systems require long life under changing environmental conditions, potential degradation of components[18], or changing requirements, such as those electronic systems designed for space missions. These types of applications represent a large portion of the emphasis of evolvable hardware work today.

Design by evolution is a proven effective method for many simple, somewhat isolated applications. Its capabilities are somewhat limited in capacity compared to more traditional, modular engineering of circuits and systems. This limitation becomes especially apparent when designing large integrated systems. The major drawback to evolvable hardware (and evolutionary computation in general) is that the user or designer frequently has very little knowledge of *how* evolution has solved a given problem. The process often results in a design that is incredibly difficult to analyze, port to other physical platforms, and/or scale. Researchers have taken different approaches to address these difficulties.

**Notation and Conventions** It is important to keep a consistent naming convention when describing many of the concepts in this field of research. Some researchers have chosen to make a distinction between evolutionary circuit design and evolvable hardware [17], indicating the latter term should be reserved for use when the EA will be run over the lifetime of a system. Others [22] have attempted to classify the landscape of evolvable hardware into more descriptive categories. This breakdown is important for understanding the possibilities within evolvable hardware. For our purposes I will adopt a less restrictive definition of evolvable hardware. I will use evolvable hardware in a more general sense to describe all of the above, to be consistent with the literature pertaining to the NASA/DoD Conference on Evolvable Hardware. However, I will distinguish between members of our population realized in software or hardware, known as extrinsic vs. intrinsic evolution. Evolution which involves the simulation of candidate solutions is known as **extrinsic evolution**. Using

software simulation, the designer can outline the design space and fitness evaluations in simulation, with only the best solutions from the final generation implemented in actual hardware. This approach is safe because unusable or poor designs are never tested on actual hardware. It is also more analytical, since the simulation can be customized to provide feedback to the designer about any aspect of the state of the evolutionary process. In addition, for circuit design, any internal node or state information can be extracted from the simulation and incorporated into the fitness evaluations. These features are not available to **intrinsic evolution**. For intrinsic evolution, all candidate solutions are implemented on real devices in their intended environment. This approach allows the algorithm to search for solutions exploiting a rich set of inputs from the environment or the specific device. Often, the results are surprising. Intrinsic evolution often converges to solutions that are particular to the specific chip used, creating designs that are neither portable nor scalable. Techniques have been devised for addressing these issues. Most involve evaluating candidate solutions across multiple FPGAs or different parts of the logic array within the same FPGA. It is believed that evolution may be taking advantage of specific properties of the silicon chip and environment (i.e. subtle capacitive effects, asynchronous logic) not typically used or understood by human designers. Because the candidate solutions may be utilizing aspects of the underlying hardware architecture in unique ways, these techniques show promise for very novel, adaptable utilization of hardware resources. Another drawback of this approach is the unpredictability of circuit response under underspecified inputs and environmental conditions once a particular design is chosen. For example, it is well known that intrinsically evolved designs often only satisfy specified behavior under low temperature ranges tolerance, and do not always degrade gracefully outside of this specified range. It is also important to note that the distinction between intrinsic and extrinsic evolution we adopt only describes where the candidate solutions are implemented, not necessarily where or when the fitness evaluations take place. For example an intrinsic solution can evaluate fitness in software by sampling signals, or by specialized testing hardware. In addition, intrinsic evolution can take place at design-time or at run-time. These distinctions do

not currently have a convenient nomenclature.

## 2.3 EHW Platforms

The limits of Evolvable Hardware have traditionally been bounded by the hardware and software technologies available to the evolutionary designer. This section describes the most popular evolvable hardware platforms that have been used, along with a number of platforms that remain unexplored. Although a number of factors are taken into consideration when choosing a platform, the main consideration is speed. Because evolutionary algorithms usually require evaluation of many generations of large populations, a great deal of emphasis is placed on having a platform that can instantiate and evaluate each individual as quickly as possible. Also, having a platform with building blocks that match the granularity of the desired application is also key.

*Extrinsic Platforms:* SPICE simulators are the ubiquitous tool of the circuit designer. They allow for design and simulation of circuits at the device(transistor, capacitor) level, based on well-known mathematical models of transistor behavior. The syntax describes and parameterizes each element, and the interconnects between those elements. This representation is well suited for the paradigm of Genetic Programming. Unlike traditional Genetic Algorithms which search a design space for candidate solutions, genetic programming searches for a *program*, or sequence of primitive design steps, to generate that design.[10] For an excellent selection of Genetic Programming applications and examples, see [14]. In the case of circuits, a Cartesian Genetic Programming approach also works particularly well. The goal of Cartesian Genetic Programming is to evolve a graph of nodes, addressed by Cartesian coordinates [13]. Since a circuit can easily be represented as a two dimensional graph, evolving the position and connection of available components allows evolution to generate a topology for the circuit. However, other methods must be used to generate parameters and/or sizing of these devices.

Extrinsic circuit evolution does have a large benefit in spite of the expensive

simulation. Because extrinsic evolution is confined to exploring well-modeled areas of electronic circuit design space, evolved designs achieve a higher levels of reliability across temperature and other environmental variations when deployed in real systems.

The main problem with evolving circuits in simulation, however, is that SPICE is very computationally intensive and inherently slow, due to the number of computations that must be performed in modeling the complex physical, chemical, and electrical properties of semiconductors. Any large design becomes impossible to attack using an evolutionary approach without immense computational resources. For that reason, most digital integrated systems are not designed at the circuit level. Because of this, Hardware Description Languages are used to abstract away much of the circuit details so that designers can focus on a functional description of a desired circuit. The key challenge is to evolve complex designs in HDL simulation, that are synthesizable for your target platform, much in the same way high level programming languages can be compiled for different instructions sets. The two most widely used HDLs are VHDL and Verilog.

*Intrinsic Platforms:* The introduction of the Xilinx 6200 series Field-Programmable Gate Arrays in 1995 gave birth to the intrinsic evolvable hardware camp. Researchers now had an evolution-friendly, relatively fast, robust device with which to experiment. Adrian Thompson is credited with much of the initial EHW work with these devices [21]. Although FPGAs are intended for use as digital devices, he found that evolution could take advantage of the underlying array of transistors to implement many *analog* functions. Through this and other early work, it was apparent that many features of the 6200 series were conducive to online hardware evolution: partial chip reconfiguration and the guarantee that a random configuration bitstreams would not damage the FPGAs. Unfortunately, these evolution-friendly FPGAs are no longer manufactured. Another line of Xilinx FPGAs, Virtex, are easily damaged by the random configuration bitstreams generated by EAs. The workaround is to model a Virtex line FPGA as a 6200 series FPGA, and perform evolution as one would on a 6200 series. Another option is to use the notion of a virtual reconfigurable device [17], which evolves a design at the HDL level where can be implemented on almost any tar-

get device. Field Programmable Analog Arrays are the FPGAs of the analog domain. They can be dynamically reconfigured to create circuits of moderate complexity. One FPAA, the Field-Programmable Transistor Array(FPTA), developed by JPL, is the first reconfigurable device designed specifically for artificial evolution. The FPTA uses an array of CMOS transistors as the building blocks, so it can be used for digital and analog circuits. While this level of granularity generates a vast design space for an evolutionary algorithm to search, scalability problems emerge. Some groups have addressed this problem by using predefined building blocks on the FPTA device [12] rather than evolving from scratch. While one could imagine using pre-defined CMOS opamps as building blocks, the workarounds and difficulty in procuring this device make COTS FPAAs a more attractive option. The Anadigm AN220E04 chip, provides dynamic reconfigurability at the granularity of the opamp and provides a C++ Application Programming Interface(API) to program the device. A similar device, the ispPAC from Lattice Semiconductors has been evaluated in [4], while another evolution-specific device, the Programmable Analog Multiplexor Array has been prototyped for intrinsic evolution from Bipolar transistor building blocks[15].Based upon a number of factors including building block granularity, bandwidth, availability, and programmability, we have chosen the Anadigm product for our evolvable hardware platform. See chapter 3.1 for a full description of our testbench.

## 2.4 Choosing a Field Programmable Analog Device

The capabilities of evolvable hardware have been limited by the availability of reconfigurable devices since the inception of the idea. Within the scope of digital circuits, a wide variety of FPGA devices are commercially available. This market is very broad, and companies (e.g. Xilinx) have been successfully developing these reconfigurable technologies for a number of years. Unfortunately, for our domain of interest, analog and mixed-signal circuit design, field programmable devices are a niche market,

and very few devices are easily obtainable. The literature (with the commendable exception of [9]) tends to describe the design application achievements which these devices support rather than devote deserving attention to comparison and evaluation of them for their evolvable hardware capabilities, and the difficulties in creating an environment or testbench to perform experiments with each of these devices. The most recent and informative evaluation of analog options was published in 2001, [20]. We considered the following devices:

- a field programmable transistor array (FPTA). JPL experiments use a device called FPTA-2 which is an implementation of an evolution-oriented reconfigurable architecture (EORA) and part of the SABLES system [20, 19]. U. of Heidelberg experiments use an apparently older version we shall refer to, in context, simply as FPTA [11].

- PAMA, Programmable Analog Multiplexor Array, developed at Catholic University of Rio de Janeiro([15])

- the Lattice Semiconductor ispPAC10 FPAA [4]

- the Anadigm FPAA family. [1]

In Table 2.4 we compare these devices on the basis of technology for programming ('Technology'), operational bandwidth, interconnection versatility ('Topo'), device resources, configuration time and relative cost. Programming technology is important because it can limit how many times a device can be reconfigured. The Anadigm AN221E04 use of SRAM is advantageous in this respect. Operating bandwidth contributes to application versatility. The specific problem domain and the building block vocabulary and representation of desired solutions for the evolutionary algorithm are device independent because mapping can generally be implemented. The device resources and, when available, their interconnection options, constitute the lowest level vocabulary and expressions of solutions. Stoica et al ([20]) would term this the lowest level of *granularity*.

| Device | Notes | Bdwdth | Topology | Resources | C-Time | Availability |
|---|---|---|---|---|---|---|
| Anadigm AN221E04 | SRAM | 500 kHz | free | 4 CABs of 2 op-amps each | 3.8 ms | commercial |
| ispPAC10 | EEPROM | 200 kHz | limited | opamps, caps, resistors | 100 ms | commercial |
| FPTA-2 | SRAM | unknown | within cell, inter-cell | MOS transis-tors | .008 ms | limited |
| PAMA | muxes | unknown | limited | BJTs, re-sistors | .08 ms | unknown |

Table 2.1: Comparison of different analog field programmable devices.

The PAMA, [15], can function as a fine-grained architecture, similar to an FPTA, or as a coarse-grained architecture when a human designer manually configures certain of its switches. It is a custom board-level design, that was created for the primary usage as an evolvable hardware platform. Therefore, it can accept random configuration and not sustain any damage. One of its unique characteristics is that, rather than using CMOS transistors like the FPTAs, it uses bipolar transistors. BJTs tend to be more suited for analog functions. Because it is a research platform, however, many of the features, availability and support for this platform are uncharacterized.

Heidelberg's FPTA ([11]) is a switched network of 256 (16 X 16) programmable CMOS transistors (half NMOS and half PMOS) arranged in a checkerboard pattern. Typically, with this FPTA, a genetic algorithm with a fixed length bit string genome directly represents a 'design' as a vector of routing bits, transistor terminal connections and channel geometry in the network. Like FPTA, FPTA-2, shown in Figure 2-2, is a 'sea of transistors' interconnected by other transistors that act as signal passing devices. It consists of an 8X8 array of cells. A cell's reconfigurable circuitry consists of 14 transistors connected through 44 switches. Each bit in the genome controls the opening of a switch. A cell also includes 3 fixed capacitors and a small number of directly configurable resistors. In assessing the FPTA as an option,

one issue is whether it offers sufficient flexibility in design expression. Though both the FPTA and FPTA-2 genome schemes technically constitute a form of topological search, the range of circuit topologies is limited because a cell can only connect to a neighboring cell. The low level nature of transistors and the density per cell contribute to flexibility. But, certainly in the analog circuit domain, humans designs are not connections of meshes of transistor arrays. It could be that the human conceptual organization of the resources into designs is key to exploiting them. In addition, this mismatch of organization makes the FPTA evolved designs difficult to comprehend.

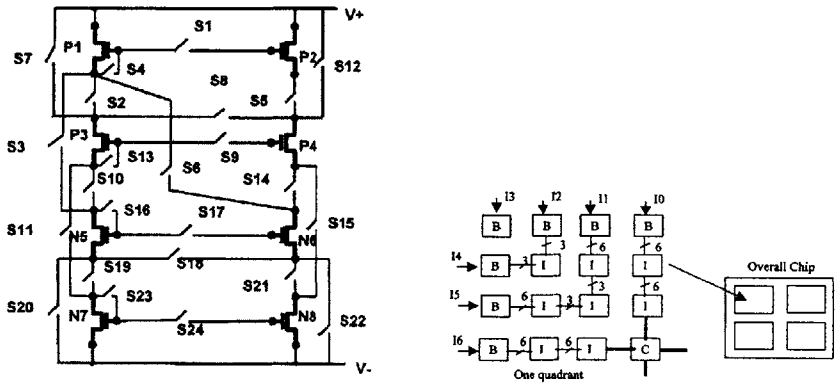


Figure 2-2: Left: FPTA-2 cell schematic showing 24 switches and 8 transistors, Right: FPTA-2 upper left quadrant of 16 cells. From [20].

Another issue one must consider is the absence of the design standard of small-signal modeling. Typically, human designers model the non-linear characteristics of transistors as linear through biasing techniques. By selecting a DC reference voltage to bias an input signal in the transistor's near-linear operating range (with respect to the range of small signal perturbations), many non-linear devices can be modeled as linear, and have stable and predictable performance characteristics. This linear behavior makes constructing approximately linear circuits out of nonlinear components tractable.

Although it is conceivable that evolution may converge to solutions with appropriate operating points, nothing in the FPTA's direct genome specification ensures this. Instead, evolution could find arbitrary biases in combination that yields a behavior

that fulfills the fitness objective to some degree. And although this individual may perform well with respect to a particular fitness evaluation, there is no guarantee this design will be robust or scalable.

This highlights a general problem in evolving circuits intrinsically, in that solutions may not be as stable across temperature and process variations of physical devices as conventional human designs. The lack of biasing is especially relevant problem for MOS devices, since their transconductances are generally lower than that of BJTs throughout their operating regions.[23]

In addition to these technical reservations, we [2] learned the price of the full FPTA-2 development system, SABLES would exceed our intended budget at around $16000. Obtaining the FPTA-2 device alone was also not a viable option, due to unresolved licensing problems. Also, additional work would have also been needed to fabricate a custom-designed printed circuit board to mount the device. Thus we continued our investigation of alternatives.

An array of op-amps, capacitors and resistors is a viable alternative to a transistor array because it offers *programmable granularity.* The array's primitive resources are versatile as building blocks. Recall that the output of an op-amp is the differential amplification of its inputs. An op-amp in closed feedback loop achieves stable linear amplification. If a capacitor is in series with the resistor in the feedback loop, a continuous integrator is obtained. Only slight topological changes will lead to the op-amp configured as an adder. Unlike transistor-level amplifiers, there is no need to bias input signals with dc offsets.

In discussion with Eduardo Torres-Jara, a graduate student at CSAIL, we considered a printed circuit board version of an op-amp array, with switchable resistors and capacitors and versatile topological interconnect switching. With additional PCBs and switches, this setup would scale to larger designs well. However, despite also being transparent to configure, a board-level reconfigurable device implementation would be slow and require other special purpose hardware to integrate into a test-bench. Therefore, we turned our consideration to the two COTS FPAAs: Anadigm AN221E04 and the Lattice Semiconductors ispPAC10.

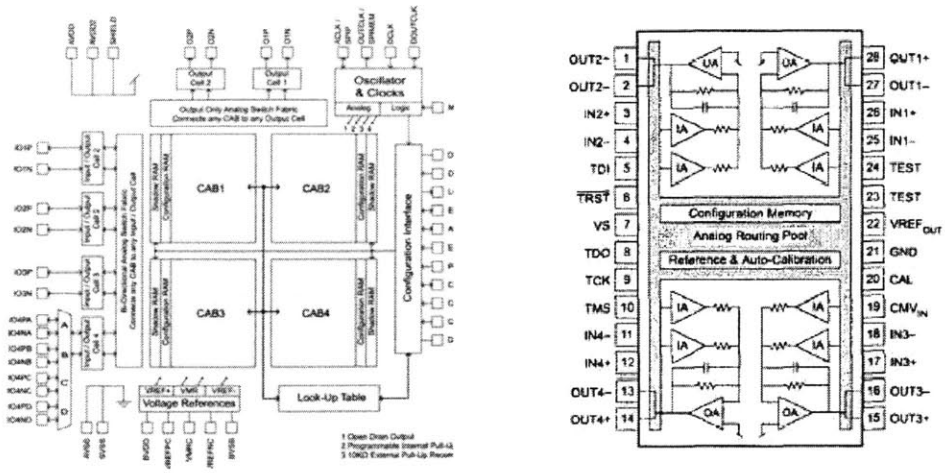


Figure 2-3: Anadigm(Left) and Lattice ispPAC10(Right) Reconfigurable Architectures

The ispPAC10, see Figure 2-3, has very limited interconnects between a small quantity of resources. It consists of 4 programmable analog modules (4 op-amps, and 8 input amplifiers total) interconnected with programmable switching networks. In [4] a non-minimal (and non-intuitive) implementation of a lead compensator was configured with the ispPAC10 that used 3 analog modules. The parameters (i.e. gains) of the compensator were optimized using simple parameter search via a GA. Greenwood has stated in [4] that the range of available capacitors was constraining. Through personal communication, [5] we learned that a partnership with a Lattice Semiconductor staff member enabled the team to write a simple conversion program to map the genome to an appropriate bitstream format. Without this partnership, configuration from EA software would be an issue. Because the Anadigm AN221E04 has more resources based on op-amps and more designer support software that might help us configure the device, we chose it. In the next section, we describe the GPIC in more detail.

## 2.5 Anadigm FPAAs

Anadigm currently offers a number of commercial FPAAs, available under the Anadigmvortex product line. These devices represents the second generation of chips of this type from Anadigm, and feature a very flexible architecture containing coarse-grained analog building blocks. At the core of this architecture is an array of these configurable analog blocks(CABs), each of which contain two opamps, 8 capacitors, a comparator, and a Successive Approximation Register(SAR), to perform 8-bit analog-to-digital conversion of signals. The chip also contains one programmable lookup table that can be used to store information about the generation of arbitrary waveforms, and is shared amongst the CABs. See the left hand block diagram of Figure 2-3. The FPAAs within the product line share a common general architecture, although they vary in their I/O capabilities, their reconfiguration capacity, and the number of CABs on a chip. All but one of the FPAAs in this line, contain a 2x2 array of CABs, which can be freely connected to one another. In addition, any signal can be routed to the I/O pins of the device through 4 programmable I/O interface blocks and two dedicated outputs, each of which can also act as filters or amplifiers.

The AN221E04 comes in an industry standard 44 lead Quad Flat-Pack(QFP) package, and is available either unmounted or mounted on a development board. The development boards provide an easy RS-232 serial port interface to a PC through the AnadigmAssistant Software. In addition, they can be easily daisy-chained to other development boards, to facilitate the generation of larger reconfigurable structures. This daisy-chaining can be achieved by attaching similar development boards with jumpers. The Anadigm GUI supports configuration of daisy-chained development boards.

The AnadigmDesigner software abstracts the configuration, switching, and wiring of low-level components in these CABs into higher-level building blocks known as Configurable Analog Modules(CAMs), such as amplifiers, filters, rectifiers, and a host of other functions. The human designer uses these CAMs as building blocks for his/her circuit. See Table 2.6 for the list of available CAMs, and Figure 2-4

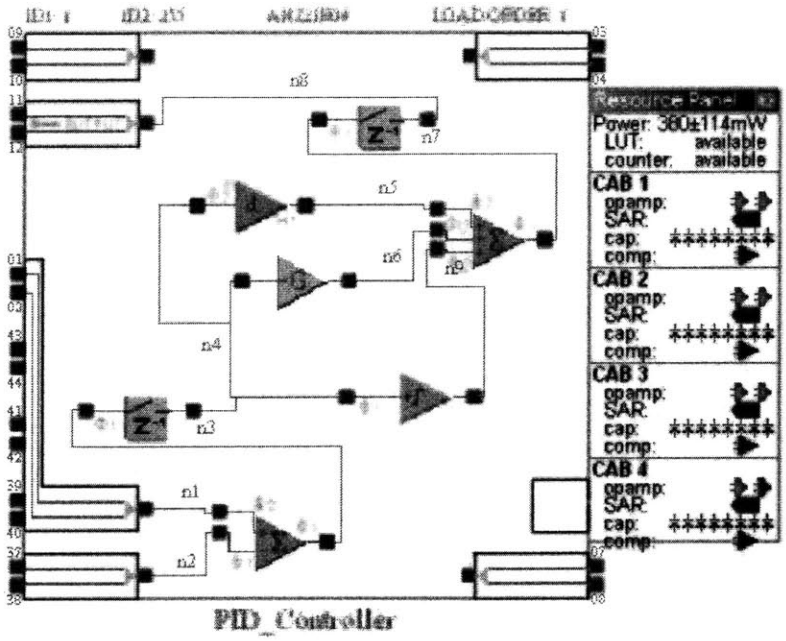


Figure 2-4: Implementation of PID controller via CAMs.(From: AnadigmDesigner2 GUI)

for a diagram of the resource allocation done for routing a signal through a simple example of a circuit implementing a PID controller. Also note that the GUI provides a convenient listing of available resources (op-amps, capacitors, comparators, and register memory elements).

Although the software provided a convenient graphic interface for designing topology and parameters for circuits, its capabilities for automating this process were limited. Initially, it seemed possible for the software package to generate C code for this automation, but this proved to be misleading. While the software allows for the generation and routing of signals between these high-level building blocks at design time, the software only allows dynamic reconfiguration of the parameters of a topology. The reconfiguration of new topologies themselves is not supported. Alternatively, this control could be obtained by writing directly to the configuration bits, although the mapping of configuration bits to switch values is not publicly available. It turns out that the predefined CAMs are an ideal set of course-grained building blocks for Genetic Programming. Section 3.1.3 describes our efforts to control this device in a portabile manner using Genetic Programming.

Another unique feature of these FPAAs is their utilization of 'switched capacitor' technology. As illustrated in Figure 2-5, these chips achieve dynamic, run-time, reconfigurability of routing and parameters through controlling switches. In addition, the configuration bits for these devices are stored in SRAM, which is more reliable than other FPAAs based on EEPROM technology. For a more detailed description of these features, see Anadigm Datasheet.

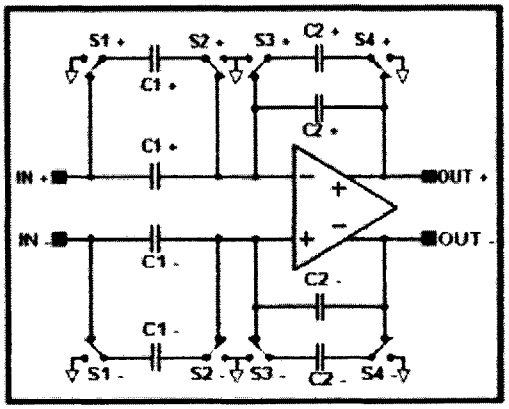


Figure 2-5: Sample Anadigm CAM, Inverting Gain block illustrating 'switched capacitor' configuration

## 2.6 Genetic Programming in Hardware

Most work on intrinsic evolvable hardware can be characterized as evolutionary search on configuration bits for reconfigurable device that results in achieving some function. However, Genetic Programming has introduced the option of evolving an indirect representation of a design, which can be much more powerful than evolving a set of configuration bits. This power comes from the inclusion of helpful abstractions from the problem domain, which improve an algorithm's effectiveness by setting up the best representations for expressing solutions.

For circuits, these representations take the form of block diagrams, program trees, math functions, Lisp expressions, and SPICE netlists. However, nearly all work done in Genetic Programming with intrinsic fitness evaluations has been done in the digital domain. Managing the mapping between designs and their corresponding chip

resources in digital reconfigurable hardware(i.e. FPGAs) is a much simpler problem than it is for analog design, since FPGAs today have millions of gates, while analog reconfigurable devices feature up to dozens of components.

Also, general purpose tools for transferring a design to the hardware resource level are not available for analog reconfigurable circuits. Therefore, a genetic representation for intrinsic analog circuit design must incorporate information about the resource limitations of the intended platform. Key to this mapping is an understanding of the actual resources with which a design can be created, and modeling the resources on a chip in such a way that the genetic representation can synthesize a valid design.

Similar to Biological systems, electronic systems are often composed of many sub-systems and components. However, circuits feature a key difference in that, at the lowest level, these designs are composed from the same resources: mostly MOS and Bipolar transistors, capacitors, resistors, and other fundamental components. There are many abstraction layers defined for human designers to manage the complexity of designing these systems. Engineers design with the respective building blocks from different levels of this abstraction hierarchy, and understanding the tradeoffs associated with describing a design using those components is critical. Using low-level resources allows a design to be fully customized down to each component, but is often infeasible for humans to design large systems at this level. By using some level of pre-defined circuit sub-structures (i.e. gates, op-amps, adders) as building blocks for more complex designs, engineers choose to tradeoff design complexity with customization. Depending on the domain of interest and end-product requirements, a designer must choose an appropriate level of representation.

In order for evolution to perform efficient and routine design of entire systems, an evolutionary algorithm must, like humans, be able to target multiple levels of abstraction and handle mapping between levels of representation. It must span the range of sufficiency for solution expression, through sufficiency for genome expression, search and genetic operators, to sufficiency for instantiation for testing and deriving fitness information. For example, it must be able to express its synthesized designs in terms of resources that are appropriate to a particular problem domain. These solutions

might have direct genome representations. Or, some additional interpretation must be added to express them as genomes. The genome representation must facilitate implementation of crossover and mutation operators. It also crucially determines the fitness landscape of the search space. Then, if the genomes cannot be directly instantiated, they must be mapped one or more times to derive a representation that can be realized physically or in simulation testing.

The EAs of Evolvable Hardware have ranged from using direct representations that are very specific to a particular device, to using indirect representations that map between multiple representations and thus isolate much of the device specific information from the EA. A prominent example of the former extreme are the projects by A. Thompson ([21] and others who used the Xilinx 6216. Fortunately, the 6216 had no configurations that could possibly short it. Thus, Thompson was able to employ a GA that used a bit vector that directly encoded the configuration vector . Although devices that have this robustness in configuration have reemerged recently, the complexity and gate count of that configuration has become intractably large. To address the issue of evolving more intelligent designs on these new FPGAs, recent work has described the idea of evolving designs for an abstraction of that hardware, described as a *Virtual Evolvable Device.* [16]

Traditionally, there have been two types of structures generated by genetic programming. Most commonly, genetic programming generates a set of instructions to execute, whose execution results in the steps for construction of a design. Alternatively, in the developmental approach, genetic programming generates a sequence of operations to perform on a 'seed', which can be manipulated using operators, to generate the structures of interest. However, recently in [10] a third representation, specific to controllers, has been described. This representation is motivated by the similarity between the block diagram representation of signal processing circuits and the program trees of genetic programming. This representation works very well for circuit design, since block diagrams are universal for describing circuits at any level of the design hierarchy. Essentially, this representation is nearly a direct-mapping between a block diagram and a program tree, with cycles in the block diagram ad-

| Building Block(CAM) | GP Function | GP Parameters |
|---|---|---|
| Inverting Gain Block | GAIN | gain value |
| Integrator | INTEGRATOR | integration constant, ref. voltage |
| Bilinear Filter | FILTER | type: {lowpass, highpass, allpass } |
| Comparator | COMPARE | reference voltage |

Table 2.2: Sample Anadigm CAMs to be used to define a GP Function Set

dressed by Automatically Defined Functions. See figure 2.6 for an illustration of this mapping.

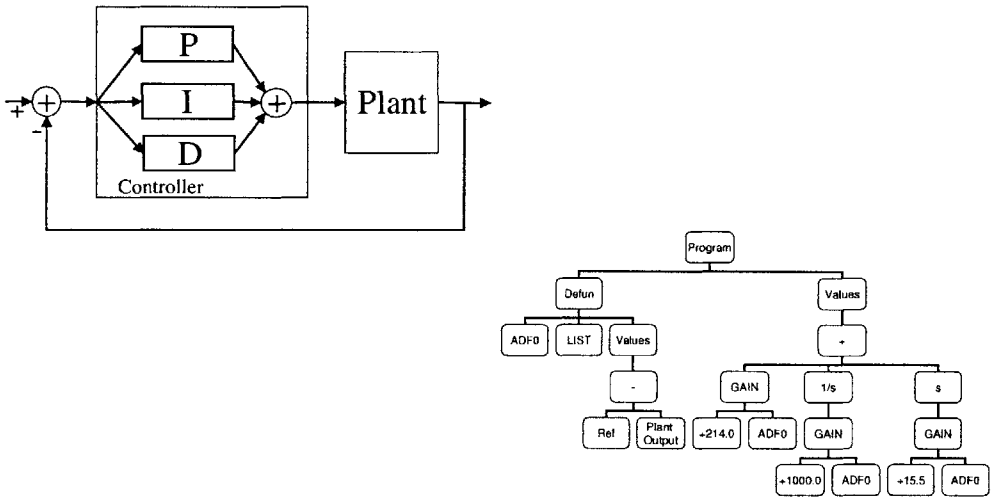


Figure 2-6: Mapping of Controller Block Diagram to Equivalent Program Tree for Genetic Programming from [10]

In Genetic Programming, a problem is typically described through defining a repertoire of functions, terminals, fitness measures, parameters, and termination criterion. For intrinsic testing, because all candidate solutions must be realizable on the platform of choice, special attention must be paid to the choice of these components of the Genetic Programming algorithm. To fulfill this requirement, one can begin defining the function set as the resources on chip, and ensuring that each design does not exceed the number of each type of resources through each of the possible genetic operators. A table of a number of sample Anadigm building blocks(CAMs) can be found in Table 2.6 along with their respective parameters.

# Chapter 3

# Results

This chapter describes the major contributions and results of this project through (1)a description of the design and implementation of our GPIC system, (2)a case study describing to what extent our current implementation achieves a specific end-to-end genetic programming run.

## 3.1 GPIC Design Summary

This section serves as a comprehensive guide to the design and implementation of GPIC. The purpose of this guide is to provide future developers and users with a description of its features, and the motivation and tradeoffs that went into these choices. In addition, each section highlights the areas of the system that were realized in the scope and time constraints of this phase of development.

### 3.1.1 System Overview

GPIC is composed of three major hardware components, and the software to configure and control these devices. The central unit of control for this system is the PC, which compiles and executes the reconfiguration algorithm, configures the FPAA via the serial port, and programs the DAQ PCI card. The DAQ has a bus which terminates at a connection block, to which we have connected BNC cables. A quick description of

the specifications of each of these components, their prices, and support information can be found in Table A.1. See also figure 3-1 for an image of the major components of the system.

Note that our design features a number of Commercial Off-the-shelf components(COTS). One of the problems in choosing COTS devices is that a systems designer must rely on specific product vendors for documentation and support throughout the lifetime of the end-system. When using very specialized components, this dependency may be problematic. Often times, certain product lines may become discontinued, or the product vendor may go out of business, leaving the systems designer or end-user with an unsupported product. In making our decisions regarding the integration of these components, we were highly motivated by the need to reduce this dependency on vendors.

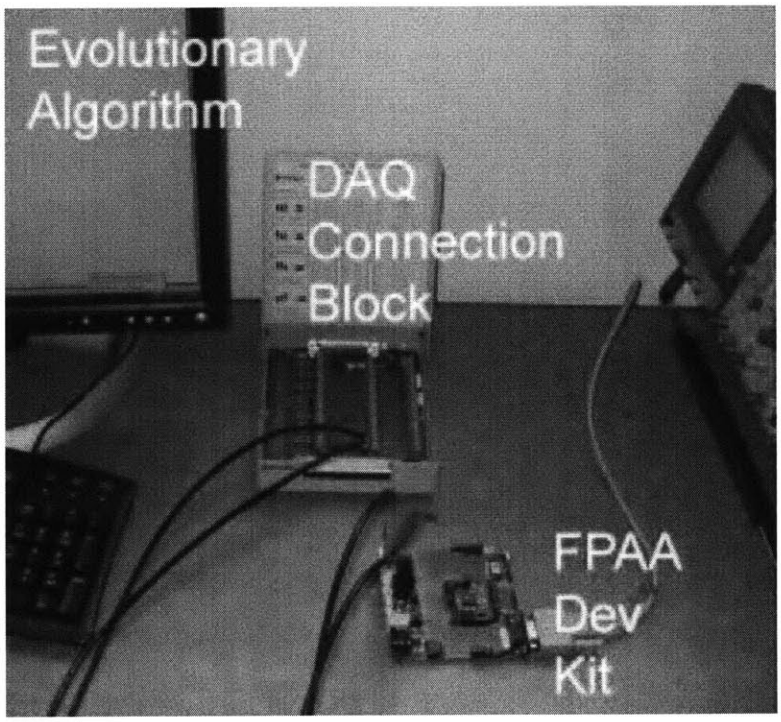


Figure 3-1: Image of Evolvable Hardware Testbench

### 3.1.2 Reconfigurable Hardware

We decided upon the Anadigm family of FPAAs for our target hardware based upon the features described in Section 2.4. Once this family of devices was chosen, we had a number of options for integrating the hardware into our test environment.

The first option we considered was populating a custom fabricated of a printed circuit board populated with one or more of the FPAAs. The advantage of this approach was that we could customize the interconnects between a desired number of FPAAs. We could also include a range of switchable building blocks for a given configuration to take advantage of. Although the design of this component would allow us a high degree of component customization in our system, it was clear this development cycle could span the length of one or more semesters. This would violate the design principles set out in chapter 1. We then began our investigation of the commercial mounting options offered by Anadigm, sacrificing another degree of independence from the product vendor. The most attractive offer for getting our testbench running quickly was to purchase the Anadigm Development Kit, which comes populated with one of their top-of-the-line FPAAs, the AN221E04 FPAA, mounted onto a Development Board. It also comes packaged with a license for the AnadigmDesigner2 software to configure this device. For $200, we were able to have a "Plug and Play" reprogrammable analog device. There was a slightly more expensive third-party board available, but we opted to go with the Anadigm solution.

### 3.1.3 Configuration of the FPAA

In selecting the Anadigm AN221E04, it was unclear how to reconfigure it from our intended platform. We also knew it would not necessarily accept random configurations, as 'evolution safe' devices could. The first alternative we considered was to ignore vendor software, and interface to some custom designed software sending configuration bitstreams to the serial port. In order to achieve our desired level of control of on-chip resources, we needed a specification sheet for how to map configuration bits to CAB switches. Since their signal routing technology is one of the key distin-

guishing features of their product line, Anadigm would not release this information. Without an explicit mapping of the configuration, our software could only attempt to repeat known good configuration streams to the chip, based upon extracting patterns from previous good configurations. These patterns were not easily discernable from a preliminary investigation of sample configurations.

Our investigation proceeded by looking at the vendor's software to achieve the desired runtime reconfigurability. Anadigm sells the Anadigm Designer-2 EDA tool which offers a simple drag-and-drop GUI for designing circuits using a vendor-defined library of building blocks. This software has a graphical interface and links up to a built-in simulator, and configuration software. The challenge was to generate these types of design and configuration events in an evolutionary algorithm. One potential solution was the AnadigmDesigner2 software's feature of 'Dynamic Configuration'. Specifically the supported 'Algorithmic Method' of code generation would allow us to auto-generate C++ code from a given circuit configuration. This code could be compiled into an executable, which could be used to perform the circuit configuration. In addition, the generated code featured functions that could be used to dynamically reconfigure the parameters of CAMs in a given circuit topology. This feature would be very conducive to online parameter tuning, (i.e. PID control), but would not satisfy the requirements that we set forth. Specifically, it was not possible to call functions to change the topology of a given circuit configuration. Recall that one of the guiding principles in this thesis is to be able to perform topological search as well as parameter search. Furthermore, the GUI package gave no indication and little documentation as to how the auto-generated C++ code was created from a given topology.

Finally, through a special agreement with Anadigm, we obtained a documentation package for how to automate events on the GUI. This package included an API description of how to automate events, as well as a description of the object models used to organize the interface between components of a given design. This package had been developed to test the GUI during product development,and was not in any product release. Generating events on the GUI was achieved through control of the

application as an Active-X object. With this package and a Microsoft C++ compiler, we were able to send designs from our GP module to the Designer-2 GUI by translating them in the GP system to a series of method calls published by the Automation API. Unfortunately, adopting this scheme of control of the chip introduced another dependency. In order to compile the code for Active-X control, we were required to port our development environment over from the GNU C++ compiler to Microsoft C++ compiler. Once we achieved Automation of the GUI, it was clear the solution proved to be conducive to demonstration, since the automation process actively displays each steps in designing a circuit on the actual GUI.

However, in order to achieve a Genetic Programming representation that could be investigated on a number of given target platforms (or future generations of Anadigm FPAAs) an abstract representation for the chip interface was needed. To address this problem, I designed a software class hierarchy that would promote portability. The key feature of this design is the use of abstract building blocks from which to create circuits. These 'Resources' would act as primitives in a Genetic Programming representation, and the tracking of these resources would be done through the abstract 'ResourceManager' class. This class could keep track of the state of resources on chip, such as the state of a switch, or the availability of an op-amp. In this model, a Genetic Programming system would not be given information about the function of particular resources. Rather, it would be presented with a finite number of each resource type, along with rules for connecting the devices to form topologies. From these, it could search for different topologies guided by the fitness metric. The most basic rule for chip resources is to describe their input/output relationships. For example, an opamp can be modeled with two inputs and one output. To genetic programming system, this device could be substituted for a MOS transistor, which can also be modeled as a two-input, one-output device. The distinction between the two would be made by their effect on the performance of the circuit during fitness evaluation.

There was also a need to represent the parameters of given resources in an equally abstract manner. Since parameters of a chip resource can range from a finite set of discrete values(i.e. a switch), to a continuous range of values(i.e. amplifier gain),

generalizing the representation was not a trivial task. In addition, the requirements on this representation are further complicated by the fact that certain parameters could affect the status of other parameters. For example, on the Anadigm AN221E04 FPAA, an IO block has a parameter describing its up as either an input or an output block. When configured as an output block, all filtering functionality, which is normally determined by resource parameters, is disabled. Therefore, Parameters were designed in such a way that they could be described by either their range of valid values, or a set of discrete valid values. For the sake of programming, these categories were further broken down into those parameters that took on integer values or 'double' precision values. The relationships between parameters is addressed through the design of a 'ParameterManager' class. A genetic programming system could update the parameters of a given resource by interfacing with the ParameterManager. A ParameterManager could be instantiated in a way that allowed it to link together related parameters of a given resource. The Anadigm version of each of these abstract 'Manager' classes was also implemented. Figure 2.6 describes this class hierarchy through an inheritance diagram.

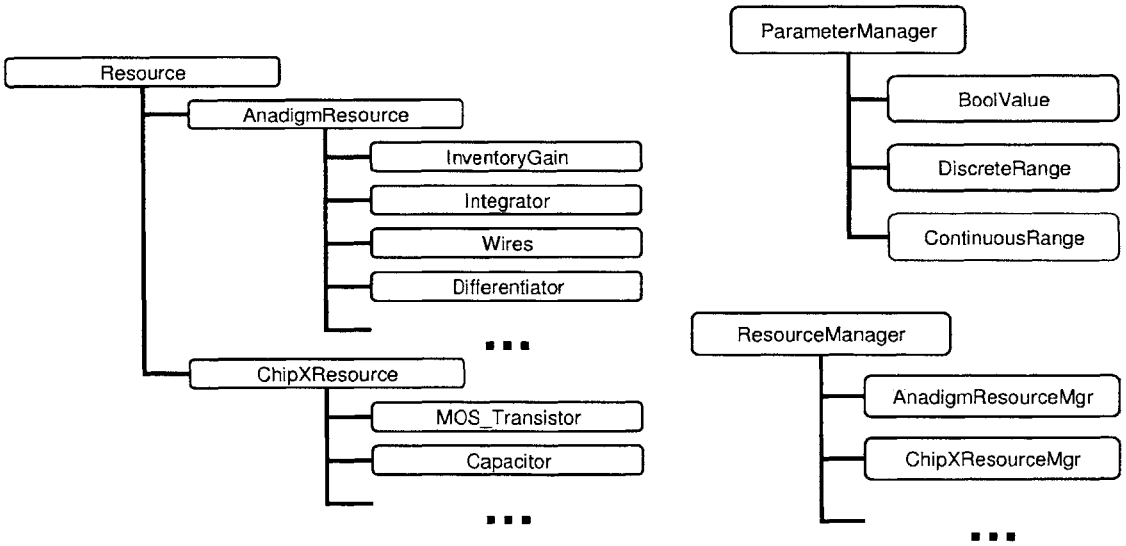


Figure 3-2: Class Hierarchy for Device-Independent Resource Abstraction.

### 3.1.4 Fitness Evaluation Module

In order to determine the fitness of circuits in a given application, the GPIC system needed a way to apply stimulus signals to candidate solutions, and sample the time and frequency domain responses corresponding to these inputs. For example, in designing a low-pass filter, one would look to apply a range of frequencies to determine the frequency response of a given circuit. To achieve our signal generation and test, two options were considered. The least expensive option would be to generate and sample signals with the sound card. This approach has a number of drawbacks rendering it unusable for many applications. Although many modern sound cards feature high grade D/A and A/D converters, the limitations of the device are many. For example, inputs to the card are DC coupled, with capacitors immediately following, the inputs. Since capacitors cannot pass DC currents, sound cards cannot sample dc values. As many of the applications we will be looking to investigate may require sampling of DC values, this approach proved to be unfit. The standard solution to generation and test of signals from a PC is the use of a Data Acquisition Card(DAQ).

#### NI Data Acquisition System

In investigating the features of Data Acquisition Cards, it was apparent that National Instruments provided the widest range of solutions in their product lines. It is important to have a solid gauge for the end application of interest when choosing a DAQ, since deciding upon features often involves trading off numerous performance metrics, such as sampling rate, accuracy of sampling, number of IOs and cost. Fortunately for our application, there was a 'low-cost Multifunction' solution, the NI PCI-6221 within the 'M Series' product line. Our intention was to test frequencies up to hundreds of kilohertz, and this solution provided that capability with a sampling rate of 833kS/s on up to 16 input channels. According to the Nyquist criterion, this rate would be sufficient for our task. For applications working in the higher frequency ranges, one might consider the 'S-series' product line. The only special requirement we had was for the DAQ to be able to generate arbitrary waveforms in addition to sampling. This

feature would be necessary when performing fitness evaluations.

In order to interface the signals generated on the DAQ with other hardware in our testbench, we were required to purchase a connection block and associated cable. Since these items, the SCB-68 Noise Rejecting Block and SHC68-68-EPM Noise Rejecting cable, were the items available for us to interface with this DAQ, the choice was simple. The combined cost of these two components totalled around $350.

To configure the DAQ in software, National Instruments provides a number of tools and drivers. For prespecified tests, the DAQ can interface with a number of available packages including Labview, NI Measurement and Automation(MAX), and VI Logger. For code-driven tests(i.e. fitness evaluations), they also provide a C API and drivers for the device. This method proved to be the most convenient way for us to integrate the control of the DAQ into our environment.

## 3.2 Fitness Evaluation Case Study

A good way to describe the characteristics and end-goal of GPIC is through a case study. This section also serves as an evaluation of the current state of its implementation

Suppose one wanted to evolve a controller for a plant with an unknown transfer function, using GPIC. Figure 3.2 illustrates the textbook feedback control loop for achieving control. See also figure 3.2 for an illustration of a unity feedback system.

This scenario would be common for systems in unknown and potentially hazardous environments, where the likelihood of having sustained hardware damage from that environment is considerable. This damage would manifest itself in changed characteristics and transfer functions of hardware. See figure 3.2 for an illustration of the typical description of these transfer functions, using Laplace transforms.

Simulating this scenario in the lab, the plant implementation could be a real plant(i.e. servo motor), a breadboard/PCB instantiation of a circuit(see Figure 3.2,

or a simulation of the transfer function in software.[1]

In the feedback loop block diagram, the FPAA would take the place of the controller and the adder, as described in figure 3.2. It would take as inputs the setpoint and feedback, and generate the input to the plant. Using Genetic Programming, we would hope to evolve a structure that would take advantage of the benefits of major and minor loop feedback.

In general, the goal of a controller is to have the output of a plant follow the input to the system. This goal would drive the evaluation of fitness of a controller. In controls, determining the performance of a given controller is done by applying a step input in the form of a square wave. By evaluating the error between the input and output, one can gauge this performance. A 'good' controller achieves performance according to metrics such as rise time, overshoot, and settling time. In our current implementation, our system has not yet defined a multi-objective fitness function which balances these metrics. At its current phase, we can custom tune a controller in the Anadigm GUI for a second-order plant, as implemented in Figure 3.2. We now have the tools and software control of all of the components in the system. The next phase of the design will involve automating this process from end-to-end.

[1] Although not currently supported, the DAQ would have to provide the input-output interface to a software-simulated plant.

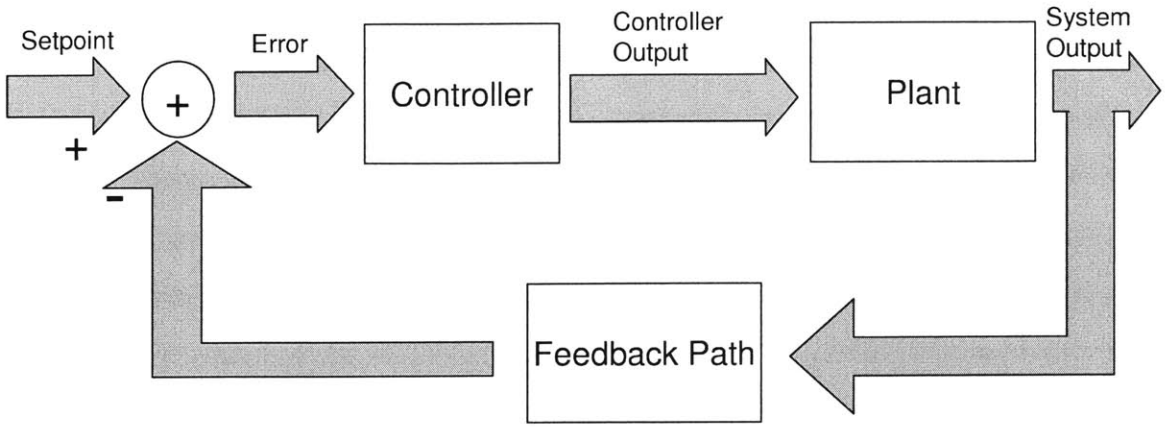


Figure 3-3: Standard Feedback System for Controlling a Plant

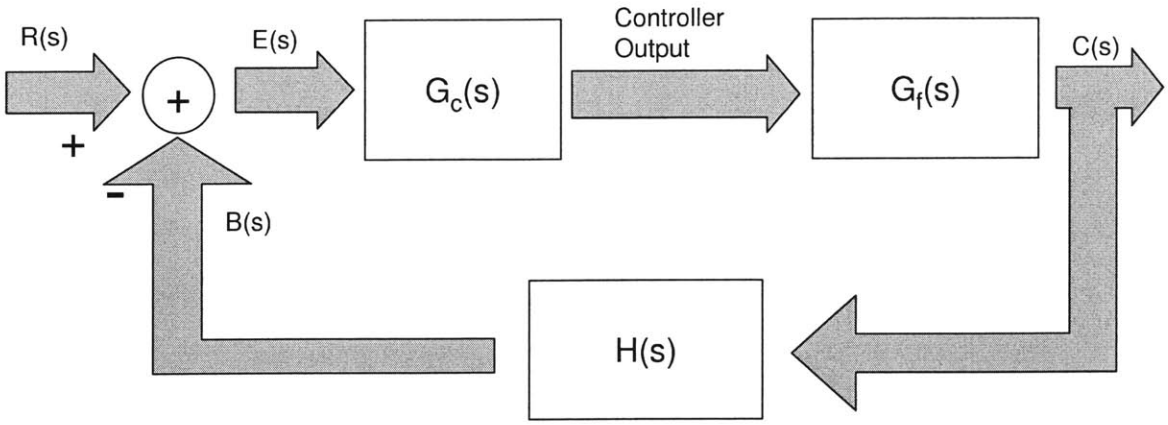


Figure 3-4: Typical Transfer Function Description of Feedback Loop using Laplace Transforms

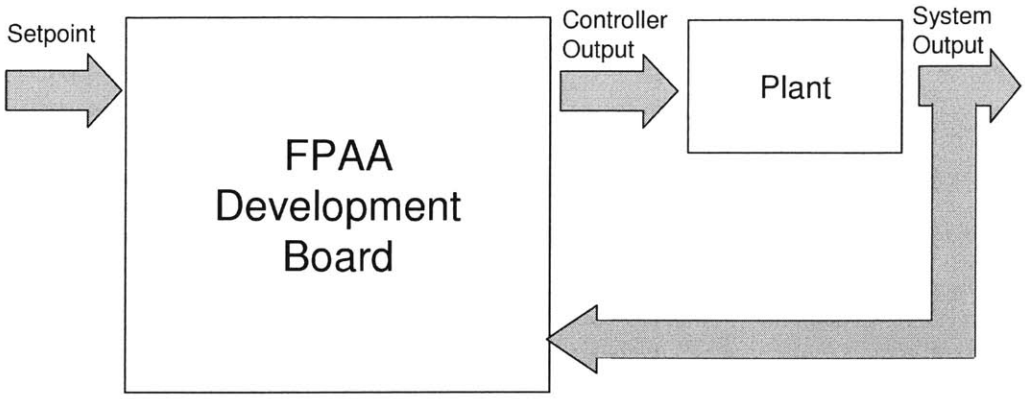


Figure 3-5: FPAA used to implement controller in Feedback Loop

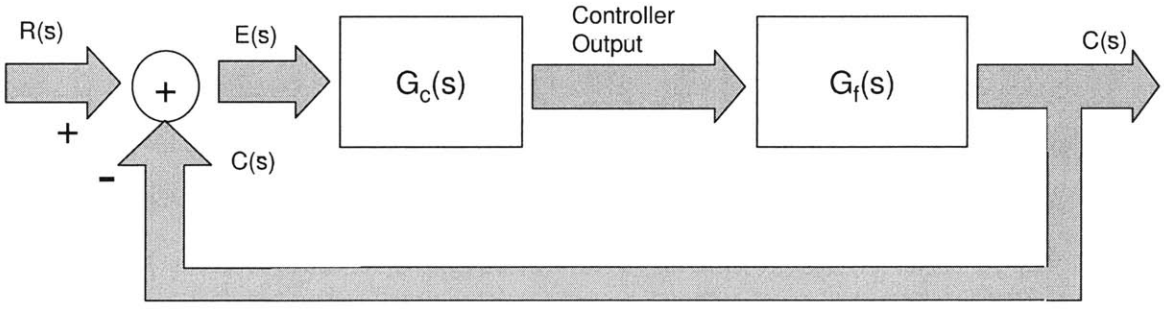


Figure 3-6: Unity feedback describes a system with no signal processing blocks in the feedback loop.

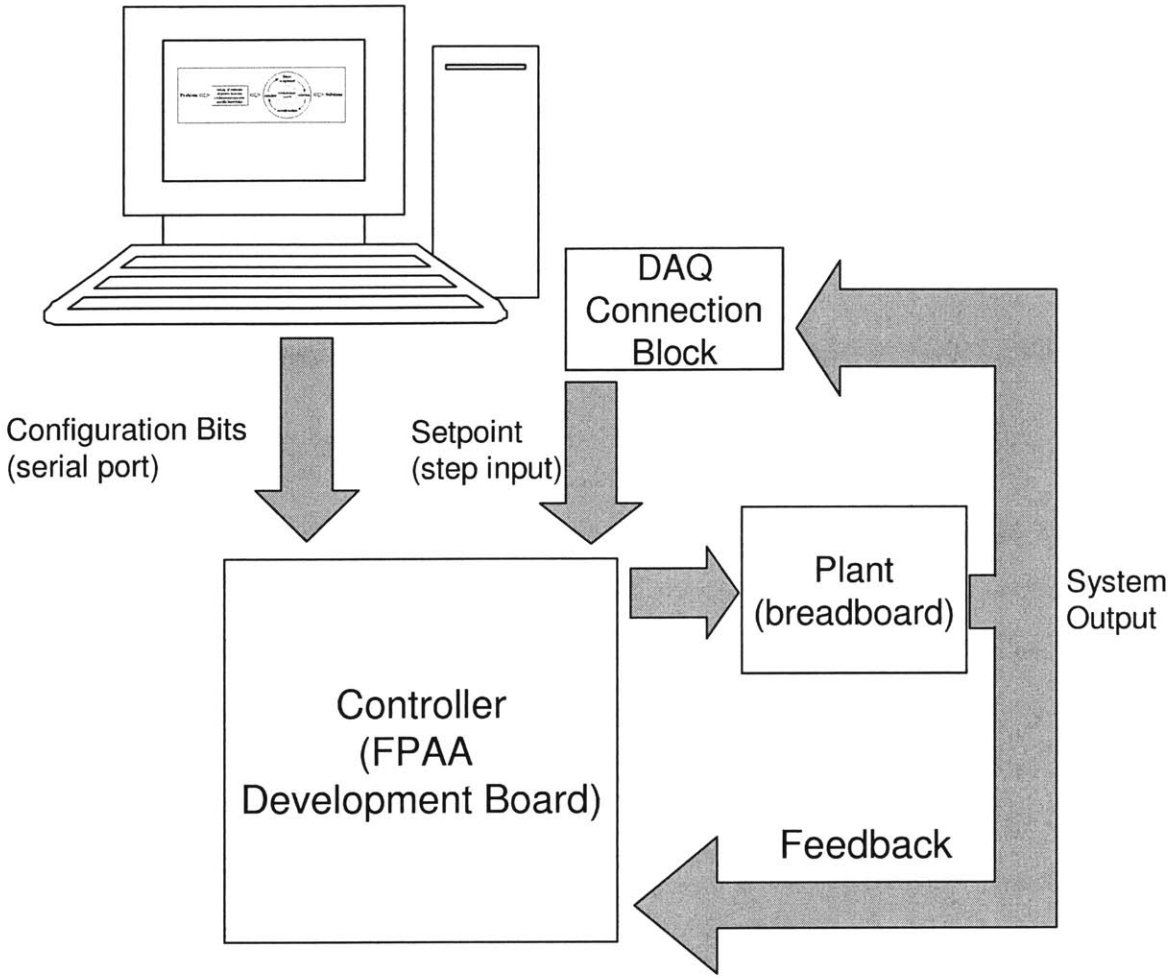


Figure 3-7: Evaluation of a controller in GPIC environment.

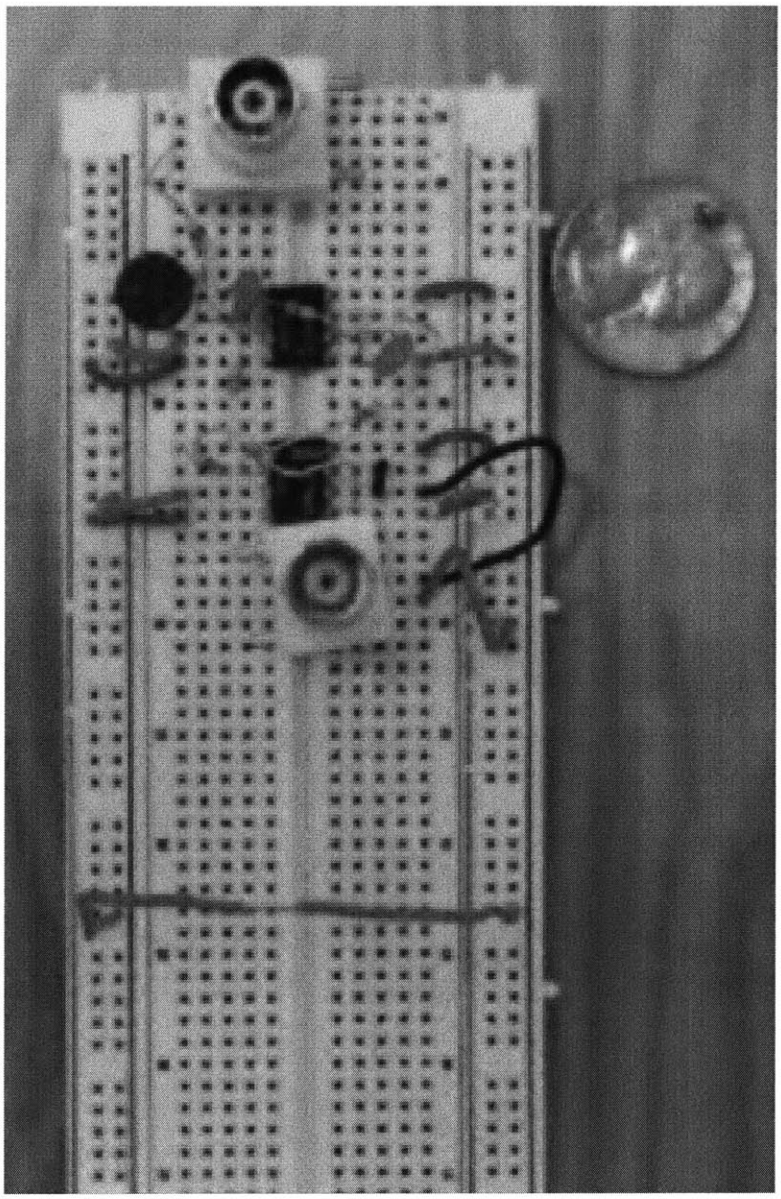

Figure 3-8: Breadboard implementation of third-order transfer function from [4] .

# Chapter 4

# Discussion and Conclusions

This section will highlight the major issues that arose in designing this evolvable hardware system, some of the lessons learned regarding evolvable hardware, and some of the unresolved research questions.

## 4.1 Merits of Evolvable Hardware

The idea of applying evolutionary algorithms to design circuits on actual hardware has been around for almost fifteen years. The ideas from this area of research have spawned two conferences dedicated solely to these ideas,and many of these ideas are featured as workshops in a handful of other conferences. Through our investigation and initial design cycle, I have developed a more clear understanding of the current trends in this field of research. From this understanding, it is clear that although there have been a number of key achievements in the field, but there are still many of the original challenges set forth from the inception of the idea. These issues are primarily related to speed of fitness evaluations, convergence of solutions, and scalability/portability of evolved designs to name a few. Although genetic programming and other developmental approaches show a great deal of potential for increasing the efficiency and scalability of evolutionary search, the path toward routine evolvable hardware is not yet in sight. The challenges of realizing even one run of intrinsic evolution have been highlighted by the history of obstacles faced during this project.

Also, since the end-goal for our evolvable hardware system is to evolve adaptive controllers, we have placed a great deal of emphasis on choosing a representation and problem domain that will allow us to achieve that end. However, the issues related to evolving feasible, safe real-time adaptive control systems have not been investigated beyond [7][6]. A more thorough investigation is needed to determine the merits of applying evolvable hardware hardware techniques to this problem, especially in relation to robotics and other autonomous systems.

## 4.2 On Intrinsic Genetic Programming for Analog Circuits

In examining the existing work on Genetic Programming and Evolvable Hardware, it is clear that the niche of the problem space most interesting to our group has been somewhat overlooked. The emphasis of existing work in Genetic Programming for analog circuit design has used circuit simulation to perform fitness evaluations on candidate solutions. As we have described in this thesis, intrinsic fitness evaluations have the potential to reduce the simulation time and complexity of a GP approach. However, all intrinsic Genetic Programming work has been restricted to the digital domain. Fortunately for digital intrinsic evolvable hardware, the problem of resource mappings is much more manageable. In the design of our system, we have addressed the problem of resource tracking, through a modular abstraction of 'Resources'. Although this choice of representation shows a great deal of promise, further investigation is needed into the feasibility of this approach.

# Chapter 5

# Future Work

Since this thesis represents the first documented initiatives at the MIT Computer Science and Artificial Intelligence Laboratory in Evolvable Hardware, this project has opened up a number of directions for research into this and related fields. These areas include additional iterations in the development of our system, design of new reconfigurable hardware platforms, characterization and development of evolutionary algorithms on GPIC. These avenues of work, and some potential solutions, are highlighted in sections below.

## 5.1 Development Cycle - Phase 2

The major contributions of this first phase of the design cycle is the design, implementation, and modular integration of components for an evolvable hardware testbench. Through this process we have increased our understanding of the requirements and problems associated with getting started in evolvable hardware. Now that these technical challenges have been understood, we can move forward with the next phase in implementation of this system. This phase will begin with addressing the number of issues have yet to be resolved in our current implementation. This development cycle may begin with the implementation of a module for synchronizing events on the Data Acquisition System. Our current implementation has the ability to generate arbitrary signals on the analog out bus, and sample inputs on any of the 18 analog

inputs, but cannot yet synchronize these two events. This synchronization is critical for fitness evaluations, since those evaluations require simultaneous signal generation and sampling.

## 5.2 The Problem of Representation

In nearly every area of artificial intelligence, the computer scientist is faced with choosing a concise and intuitive representation for the problem statement. The major difficulty lies in the expression of domain knowledge in the programming languages and data structures of today's digital computers. With Genetic Programming, this information is captured in choosing a valid syntax for designs, and by defining fitness functions and primitives in the form of function sets and terminal sets. Also, unlike traditional genetic algorithms, Genetic Programming prevents the search from exploring invalid areas of the design space through enforcement of syntax.

We have a number of choices for representing the topology and parameters of our circuits, given a chosen platform. Many of these options remain unexplored. The representation issue is further complicated by the fact that we have a prespecified number of components available on-chip from which to generate designs. This issue of choosing appropriate building blocks as primitives for circuit designs was examined in Chapter 2, and represents the beginning of a set of decisions that must be made regarding representation.

### Koza's Program Trees

A good starting point for our system would be to attempt to replicate this type of representation, with fitness evaluations in hardware. Some of the merits of this approach depend on rigorous testing during fitness evaluations, large population sizes, and parallel fitness evaluation. These features would not be achievable in our system, unless parallel fitness evaluations could be achieved on multiple chips. Understanding ways overcome these challenges would provide a great deal of benefit to the evolvable hardware community.

### Cyclic Graphs

A similar representation which could be investigated is the use of cyclic graphs to represent controller circuits. The motivations for this approach are rooted in the need for minor loop feedback in a controller design. This approach would eliminate the need for Automatically Defined Functions(ADFs) to describe this minor loop feedback. Since ADFs are correlated with large population sizes, this approach would result in a tremendous increase in the efficiency of the evolutionary algorithms.

Another simplification would be to separate topology search and parameter tuning in Genetic Programming. This part of the search could be more easily done by Hill-Climbing, especially since the realizable numerical parameters in actual hardware fall under specific predetermined bounds.

Furthermore, a design system could adopt ideas from Lamarkian evolution, by allowing the evolution system to select the best nodes on a given circuit to be the inputs or outputs. Since a cyclic representation would have no root node, a traversal of the tree could be done to find the appropriate nodes. Also, a cyclic representation would result in difficulty in performing genetic operators like crossover on different designs, since matching parts of two different designs would involve a search on the nodes of the structure(i.e. A*). A fair amount of work can be done on investigating and implementing this and other proposed representations for circuits, and tested on our evolvable hardware testbench. The key to this choice of representation is being able to perform useful mutation and crossover operations. Again, knowledge of the domain of interest can be incorporated to improve the efficiency of the representation.

## 5.3 New Reconfigurable Hardware Platforms

The development of our system highlighted the fact that there is a lack of available options for reconfigurable analog hardware. Also, the requirements and available features for existing devices had not been documented or well understood at CSAIL prior to this project. Leveraging some of the understanding that we have gained through this work, future work can be done to develop platforms that address the demands of

evolvable and reconfigurable analog hardware further. One direction to take this work is in developing printed circuit boards containing numerous reconfigurable devices, and a mixture of low-level circuit components like resistors and capacitors. Another direction would be to fully integrate a solution on a single integrated circuit.

## 5.4 Evolutionary Algorithm Evaluation and Development

As described in Chapter 4, there are a number of questions that have yet to be resolved regarding the feasibility of evolvable hardware for adaptive control. Part of solving this problem is understanding the space and memory requirements of current well-known evolutionary algorithms in hardware. A number of projects can be outlined to characterize evolutionary algorithms on our platform. In addition, evolutionary algorithms for evolving controllers remain highly uncharacterized.

# Appendix A

# Component Documentation, Support, and Notes

| Component | Specifications | Procured From | Price | Support | Notes |
|---|---|---|---|---|---|
| Dell Dimension 8400 | 3.6Ghz P4 CPU, 2GB RAM | www.dell.com | $2200+cost of monitor | 3 year on-site service | replaced heat sink, cpu fan, and motherboard May 2005 |
| FPAA Development Kit | PCB w/ FPAA and 2 Signal Conditioning Dual-opamps | www.anadigm.com | $200 w software | tech support through Anadigm | none |
| AnadigmDesigner2 | Configuration Software For Win32 Platform | www.anadigm.com | packaged with development kit | same as above | none |
| AutomationDoc | Documentation for Anadigm GUI Scripting | Anadigm Support | free | none | obtained through special agreement with Anadigm |
| NI 6221 DAQ | Multifunction DAQ w Analog Output, PCI Card | www.ni.com | $430 | tech support through NI.com | none |
| NI Connect Block and Cable | Shielded Connection Block with Cable to Interface to PCI DAQ card | www.ni.com | $350 | tech support through NI.com | none |

Table A.1: System Components

# Bibliography


[1] the PROGRAMMABLE ANALOG company Anadigm. An221e04 datasheet: Dynamically reconfigurable fpaa. http://www.anadigm.com/_doc/DS030100-U006.pdf, 2004.

[2] Ian Ferguson. Personal communication. email: October 20, 2004.

[3] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence through Simulated Evolution.* Wiley Publishing, 1966.

[4] Garrison Greenwood and David Hunter. Fault recovery in linear systems via intrinsic evolution. In Ricardo Zebulum, David Gwaltney, Gergory Hornby, Didier Keymeulen, Jason Lohn, and Adrian Stoica, editors, *Proceedings of the NASA/DoD Conference on Evolvable Hardware*, pages 115–122, Seattle, Washington, 2004. IEEE Computer Society.

[5] Garrison W. Greenwood. Personal communication. email: Jan 23, 2005.

[6] Garrison W. Greenwood. Intrinsic evolution of safe control strategies for autonomous spacecraft. *IEEE Transactions on Aerospace and Electronic Systems*, 2004.

[7] Garrison W. Greenwood. On the practicality of using intrinsic reconfiguration as a fault recovery method in analog systems. arXiv:cs.PF/0404001, 2004.

[8] John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[9] David Hunter. Some lessons learned on constructing an automated testbench for evolvable hardware experiments. In *Proceedings of the Congress on Evolutionary Computation*, pages 1808– 1812, Portland, Oregon, 2004.

[10] John R. Koza, Martin A. Kean, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV:Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, 2003.

[11] Jörg Langeheine, J. Becker, S. Fölling, Karlheinz Meier, and Johannes Schemmel. Initial studies of a new vlsa field programmable transistor array. In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, and M. Yasunaga, editors, *Evolvable Systems: From Biology to Hardware: Proceedings of 4th International Conference, ICES 2001*, volume 2210 of *Lecture Notes in Computer Science*, pages 216–, Tokyo, Japan, October 3-5 2001. Springer-Verlag.

[12] Jörg Langeheine, Martin Trefzer, Daniel Brüderle, Karlheinz Meier, and Johannes Schemmel. On the evolution of analog electronic circuits using building blocks on a cmos fpta. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors, *Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004, Proceedings, Part I*, volume 3102 of *Lecture Notes on Computer Science*, pages 1316–1327. Springer-Verlag, 2004.

[13] Julian F. Miller and Peter Thompson. Cartesian genetic programming. In *Proceedings of Third European Conference on Genetic Programming*, pages 121–132. Springer-Verlag, 2000.

[14] Una-May O'Reilly, Tina Yu, Rick L. Riolo, and Bill Worzel, editors. *Genetic Programming Theory and Practice II*, volume 8 of *Genetic Programming*, Ann Arbor, MI, USA, 13-15 May 2004. Springer.

[15] Cristina Costa Santini, Ricardo Zebulum, Marco Aurélio C. Pacheco, Marley Maria R. Vellasco, and Moisés H. Szwarcman. Pama - programmable analog multiplexer array. In *3rd NASA / DoD Workshop on Evolvable Hardware (EH 2001), 12-14 July 2001, Long Beach, CA, USA*, pages 36–43. IEEE Computer Society, 2001.

[16] Lukáš Sekanina. Virtual reconfigurable circuits for real-world applications of evolvable hardware. In *ICES*, volume 2606 of *Lecture Notes in Computer Science.* Springer, 2003.

[17] Lukáš Sekanina. *Evolvable Components: From Theory to Hardware Implementations.* SpringerVerlag, 2004.

[18] Adrian Stoica, Didier Keymeulen, Tughrul Arslan, Vu Duong, Ricardo Zebulum, Ian Ferguson, and Xin Guo. Circuit self-recovery experiments in extreme environments. In Ricardo Zebulum, David Gwaltney, Gergory Hornby, Didier Keymeulen, Jason Lohn, and Adrian Stoica, editors, *Proceedings of the NASA/DoD Conference on Evolvable Hardware*, pages 142–145, Seattle, Washington, 2004. IEEE Computer Society.

[19] Adrian Stoica, Ricardo Zebulum, M.I. Ferguson, Didier Keymeulen, and Vu Duong. Evolving circuits in seconds: Experiments with a stand-along board-level evolvable system. In Adrian Stoica, Jason Lohn, Rich Katz, Didier Keymeulen, and Ricardo Salem Zebulum, editors, *Proceedings of the NASA/DoD Conferenece on Evolvable Hardware*, pages 129–130, Alexandria, Virginia, 2002. IEEE Computer Society.

[20] Adrian Stoica, Ricardo Zebulum, Didier Keymeulen, R. Tawel, T. Daud, and A. Thakoor. Reconfigurable vlsi architectures for evolvable hardware: from experimental field programmable transistor arrays to evolution-oriented chips. *IEEE Transactions on VLSI Systems, Special Issue on Reconfigurable and Adaptive VLSI Systems*, 9(1):227–232, 2001.

[21] A. Thompson. *Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution.* Distinguished dissertation series. Springer-Verlag, 1998.

[22] Jim Torresen. An evolvable hardware tutorial. http://home.i.uio.no/jimtoer, 2004.

[23] Pedro F. Vieira, Leonardo B. Sa, Joao P. B. Botelho, and A. Mesquita. Evolutionary synthesis of analog circuits using only mos transistors. In Ricardo Zebulum, David Gwaltney, Gergory Hornby, Didier Keymeulen, Jason Lohn, and Adrian Stoica, editors, *Proceedings of the NASA/DoD Conference on Evolvable Hardware*, pages 38–45, Seattle, Washington, 2004. IEEE Computer Society.



6964-74