

BAA 97-06

Section I: Administrative

Technical topic area: Adaptive Computing Systems

Lead Organization: University of California Irvine

Type of Business: OTHER EDUCATIONAL

Team Members:

Fadi J. Kurdahi, Ph.D., University of California Irvine
Nader Bagherzadeh, Ph.D., University of California Irvine
Robert Heaton, Obsidian Technology

Dynamically Reprogrammable Architecture: A New Approach to Adaptive Computing

Technical Point of Contact

Fadi J. Kurdahi, Ph.D.
Associate Professor
Department of Electrical & Computer Engineering
University of California
Irvine, CA 92697

Tel: 714-824-8104
Fax: 714-824-2321
Email: kurdahi@ece.uci.edu

Administrative Point of Contact

Ms. Nina Crow, Grant Officer
Office of Research Administration
Sponsored Projects Administration
115 Administration Building
University of California
Irvine, CA 92697

Tel: 714-824-3572
Fax: 714-824-2094
Email: nwcrow@uci.edu

Project Duration: 7/1/1997 - 6/31/2000

<i>Total Project Expenses:</i>	\$1,182,167
<i>UC Irvine Cost Sharing:</i>	\$19,500
<i>Total Funds Requested from DARPA:</i>	\$1,162,267

Section II: Detailed Proposal Information

A. Innovative Claims

This research project proposes to break new grounds in computing by proposing a vertically integrated approach to Adaptive Computing (AC). We will research, develop, and prototype a new FPGA architecture that can be dynamically programmed *during runtime* along with a set of CAD tool support and a user-level runtime environment. A small scale prototype proof-of-concept AC chip will be designed and fabricated. The efficiency of the approach will be demonstrated on a set of defense-related target applications. Specifically, the following tasks will be addressed:

1. Dynamically Reconfigurable Architecture (DRA)

The object of this task is to develop and design a scalable, programmable architecture that can be dynamically reconfigured *during program execution* to any new circuit structure, in whole or in part, within one clock cycle. This will be accomplished by employing an SRAM-based architecture with multiple planes of configuration memory. This allows the rapid switching between hardware “contexts”. In addition, a simple controller core will be on-chip. This allows the chip to be self-controlled and removes the need of an additional off-chip controller as is the case in most FPGA-based systems. A small-scale prototype chip will be laid out and fabricated as a proof-of-concept.

2. Design tools for adaptive computing

In order to support the architecture above, we need to develop a new generation of CAD tools consisting of a library of high level components, a set of synthesis tools to map computations onto the adaptive computer, and a design environment to enable the exploration of design alternatives. Current legacy FPGA CAD tools are inadequate because they are: (1) too slow for on-line reconfiguration, and (2) do not consider interconnect (which already accounts for up to 60% of overall performance in today’s FPGA’s). Furthermore, the reconfigurability of both the computational and interconnect parts suggests new paradigms where both parts can be treated as resources dynamically allocated and deallocated on demand. Central to this approach are a suite of estimation tools that allow the gauging of design quality early in the design cycle, and corresponding architectural synthesis tools. The PIs’ previous work has shown that fast, accurate estimators coupled with layout-driven synthesis techniques can lead to mapping of computations onto FPGAs with predictable results [XK961, XK962].

3. User-level support

We will develop user-level tools to enable the DRA to run legacy code by emulating the legacy target machine. Additionally, a runtime environment will provide support for the management of tasks on the DRA, which is similar to the paging concept in memory management systems where pages are replaced on-demand as computation proceeds, where memory could represent configuration data or computation data. The main novelty in our computation paradigm is that of *evolutionary computation* of code. In essence, the code is initially compiled and run on an emulated architecture. In this architecture, the execution pipeline is reconfigured on the fly (based on the computation output) to adapt to conditional branches as well as other exceptions. As execution proceeds, on-line profiling will provide enough information for the system to “self-tune” itself by adjusting architectural parameters for improved execution of the code. Eventually, the pattern history is used to automatically migrate part of all of the code execution to a dedicated hardware configuration for optimal performance.

B. Deliverables

As part of this project, we propose the following deliverables:

1. A new, custom silicon based Dynamically Reprogrammable Architecture (DRA) will be prototyped which will allow on the fly reconfiguration of the logic in a single clock cycle. This system will be designed and simulated at the architectural level and its performance evaluated. We will investigate the different micro controller cores available on the market and select the most appropriate one to incorporate into our architecture. Specifically the following extensions to existing research [BRASS, MIT] will be provided:
 - Value of improved sector decode mapping. The benefit of an extra layer of table based programmability between global overlay control and local sector Program Selector register (PSR) will be investigated for effectiveness.
 - Practical benefits of real time conditional control of the local (sector) or global layout code via the Program Selector Registers (PSR's) will be accessed through architectural simulation.
 - The value of an on chip embedded control and management RISC core, such as the ARM(tm), will be assessed. Also an optimum strategy for interconnecting the embedded to the DRA core will be presented.
 - FPGA architectures require relatively large silicon areas. Specific architectural, circuit level and process level area reductions from existing DRA work will be demonstrated.

The usage of accurate layout area and performance estimators previously developed by the PIs will enable us to accurately gauge the metrics of the final layout for a given target fabrication technology, and to project those same metrics for future technologies as well. Since this architecture will be scalable, we fabricate and test a mid-scale version of the system as a proof of concept.

2. We will research and code prototypes of CAD tools to support the usage of the proposed architecture. Specifically, the following will be researched:
 - A set of design tools to perform the tasks of partitioning, mapping, and placement of the components to the chip.
 - An environment to support design space exploration supported by a set of powerful estimators to assess the quality of the different design alternatives vis-a-vis various metrics such as area, performance, reconfigurability, etc. The PI's previous work has shown that fast, accurate estimators are indeed feasible for FPGAs.
 - A library of components. These components differ from the libraries currently found in most CAD system in that they are at a higher level of abstraction. Typical components could be architectural constructs or even hardware implementation of runtime libraries.
3. We will research and code prototypes of system level tools to support user interaction to the adaptive computer. These tools include the following:
 - a C compiler to translate C code into a DRA configuration image,
 - a runtime environment which would run on the on-chip controller and manage the overall operation of the system.

At the present moment, we do not see any particular proprietary claims to results, prototypes or systems in the proposed research. Much of the proposed work is expected to be published in refereed journals and conferences and will be available to the public with the approval of the monitoring DARPA office.

C. Statement of Work

C.1 Introduction

The purpose of the work described in the current proposal is to develop a vertically integrated approach to adaptive computing. The project work is divided into three parts consisting of: (1) the Dynamically Reprogrammable Architecture (DRA) design, development and prototyping, (2) CAD tools for mapping computations onto the DRA, and (3) User-level tools and runtime support for the DRA.

C.2 Proposed Tasks

The development and evaluation proposed here are divided into three main Tasks that correspond to the three major components of the proposed project. Each Task consists of one or more sub-tasks.

C.2.1 Dynamically Reprogrammable Architecture (DRA)

The Dynamically Reconfigurable Architecture will be designed and implemented jointly by UCI and Obsidian Technology. Specifically, the following subtasks will be accomplished.

1. Research the existing Dynamic FPGA Architectures in both academia and industry.
2. Identify the architectural construct and features needed to accomplish the desired functionality.
3. Define the overall architecture of the DRA.
4. Develop High Level VHDL models of the DRA architecture.
5. Perform abstract functional level simulation of the VHDL model.
6. Perform low level design and simulation of the DRA chip, as outlined in Section C.3.

C.2.2 CAD Tools for Reconfigurable Computing

To support the DRA architecture, we will research, develop, and code prototypes of several CAD tools needed to efficiently map computations onto the DRA chip. Specifically, the following subtasks will be performed:

1. Based on the DRA, identify input, output, and functionality requirement for physical level design tools.
2. Research, develop and code prototype placement tool for the DRA.
3. Research, develop, and code prototype routing tool for the DRA.
4. Identify the requirements of low level synthesis tools for the DRA.
5. Research, develop, and code prototype logic partitioning tool for the DRA.
6. Research, develop, and code prototype technology mapping tool for the DRA.
7. If necessary, develop interface to existing logic synthesis tools.

8. Research, develop, and code prototype area, performance, and power estimation tools for the DRA.
9. Research, develop, and code prototype register-transfer level physical design tool for floor-planning.
10. Define user interface and desired functions of a user-directed design space exploration environment.
11. Implement a prototype design space exploration tool.
12. Identify functional specs of architectural synthesis tool for the DRA. A VHDL front end will be developed for this tool.
13. Research, develop, and code prototype architectural synthesis tool.
14. Identify and classify a set of primitive components at the gate, Register-Transfer, and architectural level.
15. Using the CAD tools developed, design and build the component repertoire. Component generators may be developed for some specific constructs such as memories.

C.2.3 User level and Runtime Environment

User level tools will provide: (1) means for users to input computational requirements and (2) efficient management of runtime. Specifically, the following tasks will be performed:

1. A C compiler front end will be developed using existing tools (gnu).
2. A C to VHDL translator will be developed to map C code to the DRA using the Synthesis tools described above.
3. Identify an existing legacy microprocessor architecture to be implemented on the DRA.
4. Develop a pipelining model for the implementation of the legacy architecture on the DRA, modified as described in Section G.3.1.
5. Simulate and test the model on standard benchmarks.
6. Identify the requirements and specific memory constructs that will be implemented as Dynamically Programmable Memory Array.
7. Identify the parameters necessary for on-line profiling.
8. Design the hardware constructs needed for on-line profiling.
9. Develop an architectural optimization policy tool based on the on-line profiling.
10. Implement the hardware necessary to perform the architectural optimization.
11. Develop a VHDL model for the dynamically reconfigurable processor.
12. Using estimation tools, simulate the VHDL model on standard benchmarks. Estimate metrics for area and performance of the execution.

13. Identify a subset of benchmarks to be executed on dedicated hardware.
14. Synthesize dedicated architecture using the CAD tools developed.
15. Using the estimation tools, simulate the execution of the benchmarks on the dedicated architectures. Estimate metrics for area and performance of the execution.
16. Compare the area/cost metrics of benchmark execution on: (1) the basic legacy architecture, (2) the improved architecture(s), and (3) the dedicated processor.

C.3 Contractors

Obsidian Technology will be responsible for a substantial portion of the low level simulation and silicon implementation of the DRA as follows:

1. Thorough study of low level current implementations in the field. At the PLU level a number of academic studies, [MIT] as well as commercial FPGA PLU architectures are expected to form a solid basis.
2. VLSI device definition. An overall device specification will be developed which is of a practical size for prototyping with a 0.6μ double metal CMOS technology. (Or better as funds permit) The definition will be embodied in a specification document as well as a pin to pin behavioral level description in VHDL.
3. VHDL simulation. The DRA will be simulated at a device level for a set of realistic system level scenarios to verify and tune the architecture.
The results of the simulation will be detailed in a functional simulation report.
4. Gate and transistor level design will be performed with commonly available design tools and MOSIS libraries. Device timing performance will be reported.
5. The DRA device will be laid out, verified and prototyped by Obsidian personnel. MOSIS IO libraries will be utilized.
A manufacturing package will be provided as one of the project deliverables which will include the layout database, the simulation data bases, bonding diagram, pin listing.
6. An evaluation PCB will be made to perform functional tests of the chip.

D. Description of the results, products, and expected technology transfer path

D.1 Overview

The emergence of highly-flexible, field-programmable hardware, and the rapidly increasing density of devices on a chip present unique opportunities for a new era in computing. Adaptive Computing is not only a new hardware implementation for processing information, but rather a new computation paradigm. This new paradigm promises not only faster and more efficient processing of key applications (such as image processing, compression, encryption, etc..), but also a faster transition of new algorithms into an existing legacy of systems, and therefore significant savings in upgrade costs.

The work performed under this project is expected to generate significant contributions in several areas. We will investigate the feasibility of a Dynamically Reprogrammable Architecture (DRA) through the design and implementation of a chip. Along with the chip, a set of CAD tools for mapping computations will be researched and prototyped. Representative applications will be tested on the vertically integrated testbed and results will be studied.

D.2 Research Results.

D.2.1 The Dynamically Reprogrammable Architecture

The object of this work is to develop a silicon architecture for a Dynamically Reconfigurable Architecture (DRA), which is a large, scalable, processing unit that can be dynamically reconfigured to any new circuit structure, in whole or in part, within one clock cycle. The intent is provide a generic processing element designed to be configured on the fly such that the following goals can be achieved:

- Just in time reconfiguration will allow immediate re-use of processing elements allowing hardware implementation of user algorithms with the minimum of silicon area.
- Used as a co-processor the device will have the ability to support multiple processing threads, each performing an entirely different function.
- Each silicon device will be arranged such that they may form a homogeneous two dimensional array using chip on substrate packaging technologies. The intent being to provide indefinite extensibility with the minimum loss of performance.

The proposed Dynamically Reprogrammable Architecture (DRA) is enhanced with a general microprocessor in addition to the reconfigurable logic incorporated into the design. The reprogrammable part of the system could be used to function with various hardware “personalities” which are capable of numerous operations that could be very distinct from each other. For instance, at one time, the DRA might be acting as a GPS locator to guide the projectile in flight to the pre-assigned coordinate under the satellite-supplied data. Once the projectile reaches the coordinate and able to lock onto its target using radar or some other mechanism, then the device could be reconfigured to be a target analyzer. It can be reprogrammed as an image analyzer to pattern-match the current image to the database of pre-programmed image data to find the best target. While this example shows very basic reutilization of a fixed resource, the very programmable characteristics of DRA enables the system to be self-modifying on-the-fly. These concepts are also beneficial to the new philosophy within DoD of consolidating military hardware among the different

branches. A “generic” version of military hardware can be personalized to each branch’s specific needs thereby greatly reducing the NRE costs.

Conventional implementation would require signals to go off-chip for the increasing the reconfiguration time and the communication delay, and limiting the routing/interconnection options. However, by incorporating the heterogeneous components such as a general microprocessor and FPGA on-chip, many of these problems disappear, and new benefits assert themselves. The reconfigurability of FPGA has been well documented [MIT, BRASS, BYU, BYU2, BYU3, PH95] in various roles. By having a conventional microprocessor on-board to do generic computation and to aid in FPGA reconfiguration, the system is fully capable of self-servicing without any other exotic modules that is unfamiliar to both programmers and hardware designers.

The basic processing elements are familiar from RAM based FPGA architectures. These can be divided into two categories: Routing and programmable logic modules. The key innovation in this architecture is that instead of one RAM bit per program option 2^n will be provided and will typically be 4, 8, or 16 bits. Hence in effect the device becomes an FPGA loaded with up to 2^n programs. To this structure the following innovations are added:

- At each routing point and programmable logic module (PLM) a 2^n select or selects which RAM bit will be utilized.
- The device will be split up into sectors each with an independent Program Selector Register (PSR) so that the device may be partially reprogrammed.
- Program loading and sequencing may ultimately be controlled by a small local embedded processor such as the ARM with a small quantity of memory as necessary. Such a device (a CPU with a built in DRA) is a powerful and flexible device as a field programmable ASIC in it’s own right in a commercial environment.
- A goal will be that unused program storage sectors will be available to the architecture as general purpose storage RAM.
- It is intended that Fast, low voltage swing buffer IOs will be a programmable IO option such that fast inter chip communication can be achieved within an array of DRA devices when mounted either directly on a substrate or as packaged devices on a printed circuit board.
- The programmable matrix nature of the architecture may make wafer scale implementations a practical long term goal.

The ultimate aim of the above is to achieve fast ”soft hardware” which can support processing at hardware speeds while achieving the programmability and silicon re-use normally associated with software and CPU implementations.

D.2.2 CAD tools for the DRA

In order to support the architecture above, we need to develop a new generation of CAD tools. In particular, we need to develop a library of high level components, a set of synthesis tools to map computations onto the adaptive computer, and a design environment to enable the exploration of design alternatives. While such tools have already been available for today’s FPGA’s, they fall short of achieving high performance due to the fact that they are: (1) too slow for on-line reconfiguration, and (2) do not consider interconnect which already accounts for up to 60% of overall performance in today’s FPGA’s. Thus, such tools can lead to mappings with unpredictable

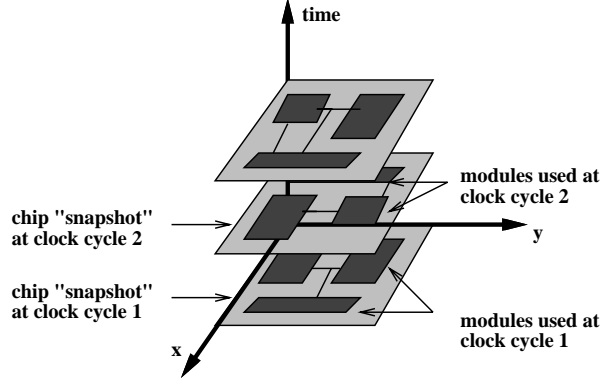


Figure 1: Temporal-Spatial Placement Approach to High Level Synthesis

performance e. Furthermore, the reconfigurability of both the computational and interconnect parts suggests new paradigms where both parts can be treated as resources dynamically allocated and deallocated on demand, thus achieving optimal utilization of available resources. Central to this approach are a suite of estimation tools that allow the gauging of design quality early in the design cycle, and corresponding architectural synthesis tools. The PIs' previous work has shown that fast, accurate estimators are indeed feasible for FPGAs given a high level description [XK961]. Using these tools, it is possible to perform architectural level mapping of computations onto FPGA's with predictable results, as has been shown in initial work by the PIs [XK962].

While prior and current research paves the way to producing better mappings of computation onto today's statically programmable FPGA, there is virtually no research work in CAD tools for dynamic architectures (i.e. those that can be reconfigured during program execution). The proposed CAD tools will be developed from the bottom up with that concept in mind. One way to visualize the desired functionality of such tools is to think of a computation as a succession of DRA configurations one for each time step (a time step being one or more clock cycles) as illustrated in Figure 1. Thus, at each step, the necessary resources (functional units, storage *and interconnect*) are instantiated and "swapped" into the current context to execute the operations scheduled for that particular time step. The job of the synthesis tools would be to optimally schedule the computation onto the time-space cube shown in Figure 1 so as to satisfy some cost function (or alternatively some constraint) on area and/or performance. Of course, the tools must take into consideration the interconnect cost/performance.

D.2.3 Computational Models and System-level Management

The ultimate goal of adaptive computing is to support high level specification input. This means that one should be able to compile high level code into the machine as if it were a "hard-wired" processor (at least initially), or even run legacy code by emulating the legacy target machine. Additionally, there should be support for the management of tasks on the DRA so as to achieve maximal utilization of available resources. Based on the Dynamically Reprogrammable Architecture, the management of tasks is similar to the paging concept in memory management systems where pages are replaced on-demand as computation proceeds. In the case of DRA, the SRAM configuration will be brought in as program execution proceeds except that in this case, memory could represent configuration data or computation data.

We identify two models of computation vis-a-vis the DRA. The first one is based on the legacy of existing architecture and is referred to as the Fully Reconfigurable Microprocessor Technology (FRMT). FRMT design relies on partitioning the microprocessor to two distinctive areas: (a) the

reconfigurable core area and (b) the attached FPGA block. The former section hereafter called RC for reconfigurable core is consist of a core CPU architecture optimized with the basic functions that are needed for an advanced superscalar microprocessor—note that this optimized core can be changed for different situations (i.e., different core technologies). For instance, one may desire to have a VLIW optimized core as oppose to a superscalar. The latter partition in the proposed microprocessor is used to optimized specialized instructions for improving the performance, similar to a coprocessor. In Section G.3.1.1, we propose a revolutionary new approach to designing the basic pipeline execution stages of the RC using a number of reconfigurable elements (RE). This is called the Superscalar Dynamic Reconfigurable Processor (SDRP). In this model, the execution pipeline is reconfigured on the fly, during program execution to adapt to conditional branches. This avoids pipeline stalls and achieves maximum utilization of the pipeline stages.

While the proposed technique ensures efficient execution of code, it does not consider the memory access issues, which can be a major bottleneck in performance. A key component in any general purpose processor is a good memory architecture. We extend the notion of reconfigurable computing to include reconfigurable memory. Memory which is reconfigurable can adapt to a program’s need for improved performance. For example, instead of having fixed sizes for the instruction cache, data cache, register files, branch target buffer, and branch prediction information, the application can adjust the sizes of these structures at run-time to maximize performance. Although FPGA’s may be well suited to dynamically reprogram computational units, they are very inefficient in providing a reprogrammable memory architecture. A different type of structure is required to provide the flexibility and density objectives of reconfigurable memory.

We propose to incorporate a Dynamically Reprogrammable Memory Array (DPMA) structure. The DPMA can be reconfigured on the fly to behave as different types of memory (such as cache, lookup table, instruction buffer) whose size can also be varied during execution. Thus, during code execution, a runtime profiler (which can be either built into the emulated architecture or as software) monitors different parts of the architecture and collects information that is used by another block, called the architectural optimizer to reconfigure the DPMA (and possibly other parts of the architecture as well) to improve performance. The DPMA is further discussed in Section G.3.1.3.

Ideally, a given computation would run fastest on a dedicated architecture. However, the efficiency would depend on several factors such as the type of data, the memory access patterns and so on. Many of these factors can only be determined at runtime. Furthermore, the large amount of legacy code also dictates running computation on a legacy processor. Thus, we adopt a hybrid execution model this project called evolutionary computation. Our approach brings together several models and concepts that could only be feasible now because of the DRA.

Evolutionary Computation is depicted in Figure 2. Essentially, the input code is compiled into a legacy processor which is emulated on the DRA using the FRMT. As execution progresses, performance is further optimized by fine-tuning the architecture as described above. Eventually, code execution can migrate to a dedicated architecture which can be synthesized using the CAD tools developed as part of this project. This is further discussed in Section G.3.3.

D.3 Products

Dynamically reconfigurable devices are a natural evolution of today’s FPGAs. The capability of a chip to be reconfigured on the fly represents an immense potential for a wide array of products. Imagine a subscription-based agreement with a hardware vendor whereby new hardware upgrades can be downloaded over the network, just like what is done nowadays with modem vendors to upgrade modems to new standards. Except that this can be done on a much wider scale. Not only

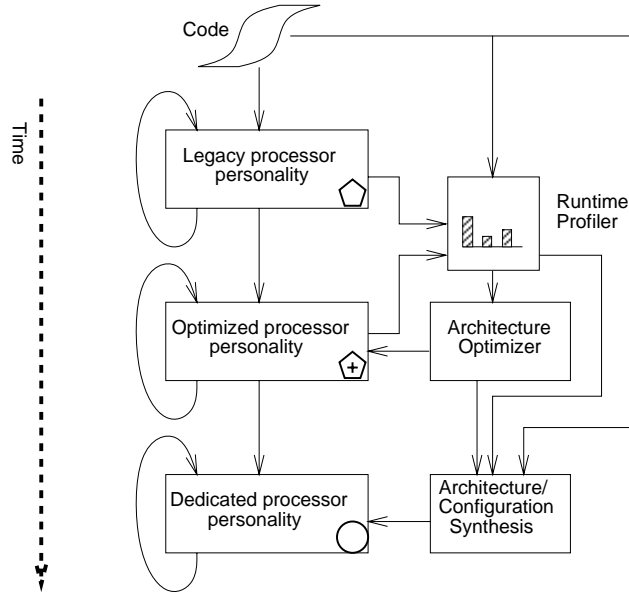


Figure 2: Evolutionary Computation

would a chip design and fabrication take less time, but hardware bug fixes can also be downloaded instead of a costly physical swap, thus allowing a faster time to market of a given product.

D.4 Technology Transfer

The PIs are committed to using the novel and traditional means of technology transfer. These include but are not limited to: reports to DARPA, journal and conference publications, technical reports made available on the WWW.

D.5 Proprietary Claims

There are no unusual proprietary claims to results, prototypes or systems in the proposed research. As our objective is to deliver high-performance systems, we may include in our software vendor proprietary information and methods that will remain the exclusive property of the vendor.

E. Identification of Conflicts of Interest

Not Applicable.

F. Cost, schedule and milestones

F.1 Breakdown of Cost Estimates

	<i>Year 1</i>	<i>Year 2</i>	<i>Year 3</i>	Total
Personnel	100,038	105,537	111,321	316,896
Equipment	20,000	20,000	0	40,000
Travel	10,000	10,000	10,000	30,000
M&S	6,000	6,000	7,000	19,000
Other	39,6720	43,639	148,003	250,314
Subcontracts	111,000	90,500	98,500	300,000
Indirect Cost	70,378	60,647	113,932	244,957
Year Total	357,088	336,323	488,756	1,182,167

NOTES: Indirect cost of 49.9% does not apply to equipment and tuition, and is applied to the first \$25,000 of subcontracts. This budget includes a subcontract totalling \$300,000 support to Obsidian. In addition, \$100,000 is requested during the third year to fund the fabrication of the DRA chip through MOSIS, and its subsequent testing.

UCI will cost share this proposal with the amount of \$19,500. Thus, the total amount requested from DARPA is \$1,162,267 over three years.

F.2 Schedule Summary

1. Year one:

- research and evaluation of DR architecture
- identification of the on-chip microcontroller
- identification and classification of key library components
- prototype C to dataflow compiler

2. Year two:

- circuit design of a small scale DRA device.
- research and evaluation of CAD synthesis tools for DRA

3. Year three:

- research and develop a prototype runtime environment
- research and develop a prototype design space exploration environment
- test benchmark application (simulation only)
- fabricate a small scale DRA chip using MOSIS

G. Technical Rationale

G.1 Dynamically Reprogrammable Architecture (DRA): An intelligent RAM Architecture

G.1.1 Overview

The object of this work is to develop a silicon architecture for a large, scalable, processing unit that can be dynamically reconfigured to any new circuit structure, in whole or in part, within one clock cycle. The intent is provide a generic processing element designed to be configured on the fly such that the following goals can be achieved:

1. Just in time reconfiguration will allow immediate re-use of processing elements allowing hardware implementation of user algorithms with the minimum of silicon area.
2. Local reconfiguration circuitry will provide the option for real time conditional selection of configuration code. This is a key requirement for adaptive systems.
3. Used as a co-processor the device will have the ability to support multiple processing threads, each performing an entirely different function.
4. Each silicon device will be arranged such that they may form a homogeneous two dimensional array using chip on substrate packaging technologies. The intent being to provide indefinite extensibility with the minimum loss of performance.
5. The programmable matrix nature of the architecture may make wafer scale implementations a practical long term goal.

G.1.2 VLSI Architecture of the DRA

The basic processing elements are familiar from RAM based FPGA architectures. These can be divided into two categories: Routing and programmable logic modules. The key differentiating factor in these architectures is that instead of one RAM bit per program option 2^n will be provided and will typically be 4, 8, or 16 bits. Hence in effect the device becomes an FPGA loaded with up to 2^n programs. To this structure the following innovations are added: (See Figure 3)

1. At each routing point and programmable logic unit (PLU) a 2^n selector selects which RAM bit will be utilized.
2. The device will be split up into sectors each with an independent Program Selector Register (Mapper) so that the device may be partially reprogrammed.
3. The Mapper contents will be indirected through a mapping function which will provide mapping between a global context vector, a set of conditional inputs. The output of the mapping function will be n address lines which will directly address one of the alternative sector codes. Additionally the Mapper will be able to provide local selection of the clock for the local context selectors.

The conditional inputs to the Mapper are taken from the north crossbar of the cell. Hence by an appropriate coding of active crossbar coding the conditional inputs may be taken from the local PLU or any other PLU within the device.

The crossbar matrix is of a conventional RAM based FPGA type except for the n contexts available to it.

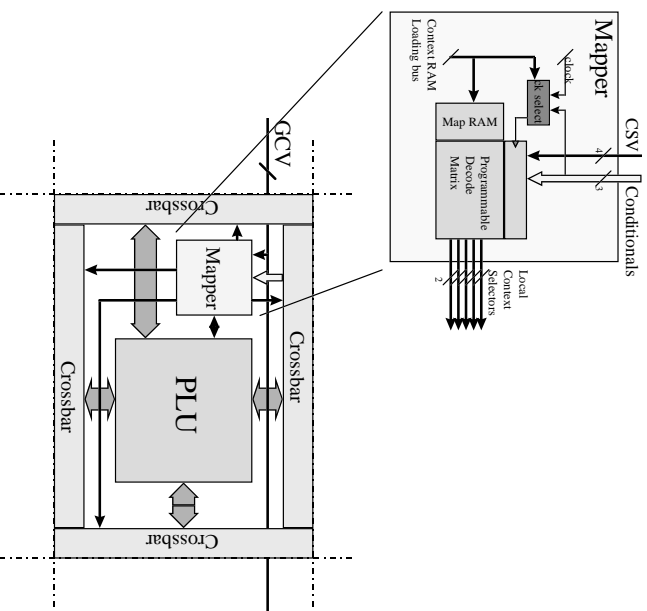


Figure 3: Sector Cell Arrangement.

4. Global, long, and inter-PLU wires form part of the crossbar fields to give a range of efficiencies for inter PLU connections per Xilinx RAM based FPGA architectures.
 5. Program loading and sequencing may ultimately be controlled by a small local embedded processor such as the ARM with a small quantity of memory as necessary. Such a device (a CPU with a built in DRA) is a powerful and flexible device as a field programmable ASIC in it's own right in a commercial environment.
 6. A goal will be that unused program storage sectors will be available to the architecture as general purpose storage RAM.
 7. It is intended that Fast, low voltage swing buffer IOs will be a programmable IO option such that fast inter chip communication can be achieved within an array of DRA devices when mounted either directly on a substrate or as packaged devices on a printed circuit board. (See Figure 4)
- Signal bond pads are placed such that inter chip connections are very short and of low capacitance to give very low propagation delay and dense silicon on substrate implementation. Hence large arrays of DRA devices are expected to be practical.
- The ultimate aim of the above is to achieve fast "soft hardware" which can support processing at hardware speeds while achieving the programmability and silicon re-use normally associated with software and CPU implementations.

G.1.3 VLSI Development Steps

The following are elements of the silicon development plan:

1. Thorough study of current research work. At the PLU level a number of academic studies, [MIT] as well as commercial FPGA PLU architectures have a lot to offer here.

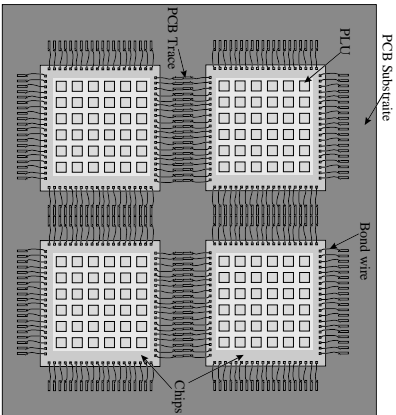


Figure 4: Chip on board matrix connection arrangement.

2. VLSI device definition. An overall device specification will be developed which is of a practical size for prototyping with a 0.6μ double metal CMOS technology. (Or better as funds permit)
3. VHDL simulation. The DRA will be simulated at a device level for a set of realistic system level scenarios to verify and tune the architecture.
4. The DRA device with be laid out, verified and prototyped.
5. An evaluation PCB will be made to test the performance of the chip.

Obsidian Technology will be providing transistor level design and layout.

G.2 CAD tools

There is already an existing legacy of CAD tools to support lower levels of FPPGA design. These tools include logic synthesis, partitioning, mapping, placement and routing. However, these tools target the existing FPGA architectures and not the proposed one. Thus, it is necessary to investigate their applicability to the DRA and either develop new point tools or adapt existing tools.

In order to bridge the gap between the user level specification and the logic level input needed to the CAD tools, there is a need to incorporate architectural synthesis techniques into our design flow. This is particularly important if configuration images must be generated at compile time or even on-the-fly at runtime. The main difficulties in utilizing current work on synthesis tools nowadays are the following

1. that they do not consider the impact of layout effect when synthesizing the designs. This is unacceptable when a design is targeted to programmable devices since programmable interconnect is expensive in both area and delay. In a highly flexible programmable devices, interconnect is expected to be even more significant in impact. Furthermore, technology projections indicate that at 0.35μ and below wiring will significantly dominate performance, even in full-custom logic. The impact will be even more severe in programmable logic where interconnect includes switching devices.
2. they are too slow for on-line reconfiguration where new architectures may have to be generated on-the-fly. The hierarchy of synthesis tools that must be used to go from code to configuration bits can easily take a few hours on a fast computer.

3. they are targeted towards *static architectures* i.e. architectures whose structure does not change during execution. In today's architectures, processing and storage units are fixed in functionality and interconnection patterns are predefined.

To combat the first problem, we must develop means of accurately predicting the design metrics (such as area, performance, and power) early in the design cycle and to incorporate these estimates in the synthesis process. Towards that end, the current research work at UC Irvine has resulted in highly accurate, runtime efficient estimators for SRAM-based FPGAs. In addition, we are currently developing synthesis techniques including scheduling, allocation, and binding techniques to utilize these estimators [6,7]. This methodology can be adapted to the proposed architecture and will allow us to leapfrog the research to generate acceptable performance tools in relatively short time.

The second problem occurs because most synthesis tools used in today's FPGAs operate at the gate level (even when using architectural synthesis tools). It is well known that logic synthesis tools are time consuming. The physical level design tools (placement and routing) are also runtime inefficient, especially when dealing with FPGAs where the problems are further compounded by having to select different routing resources for each connection. Our proposed approach is to operate at a higher level of abstraction which is the architectural level. Once an architecture is obtained, the floorplan footprint is assembled from pre-designed macroblocks whose configuration info resides in a library. In order to achieve maximum layout efficiency and to avoid "white space" in the footprint, it is possible to keep a number of alternative realizations of each block and have the assembly tool pick the most appropriate realization achieving the final design goals such as area efficiency or performance optimization. Our architectural estimation tool is configured to work at the architectural level using this physical design model. Benchmarking has shown that this approach can achieve orders of magnitude of additional runtime efficiency as compared to synthesis at the gate (or CLB) level.

The third problem is due to the fact that the proposed architecture is dynamically reconfigurable. This means that both the functional units and the interconnect, as well as the storage can be reconfigured during runtime. While current architectural synthesis techniques can still be used to synthesize architectures that can be mapped onto the DRA, such techniques fall short of taking full advantage of the dynamic nature of the chip since the generated architecture is static. In order to optimally utilize the available resources and capabilities, we must research and develop new synthesis techniques. Such techniques must be capable of producing dynamic architecture. For example, given a conditional branch in the execution, the configuration memory may contain two contexts, each corresponding to the logic needed to execute one of the branches. During execution, the branch condition can dynamically reconfigure the logic by switching to the configuration context corresponding to that branch. In addition, some situations may dictate that the input computation may run most efficiently if a new component (which does not exist in a library) can be instantiated on-the-fly and used in the architecture. Indeed, our experience with DSP computations indicates that one can identify some recurring patterns in the computation graph which when implemented in dedicated components can improve the overall computation speed and area efficiency [RK92]. Another situation arises in routing data from one point to another. Since routing is reconfigurable, one can treat interconnect as just another set of resources that can be allocated and deallocated on demand. When data is to be moved from one point to another, an appropriate route is instantiated beforehand. Upon completion of the move, the resources used can be deallocated and are available for other transfers. Note that the routing of a particular move may vary from one iteration of execution to another depending on the available routing resources at different times. In a way, this bears some resemblance to the way routing is done in communication networks.

Our approach to developing the CAD tools needed for this task will be as follows: First we will

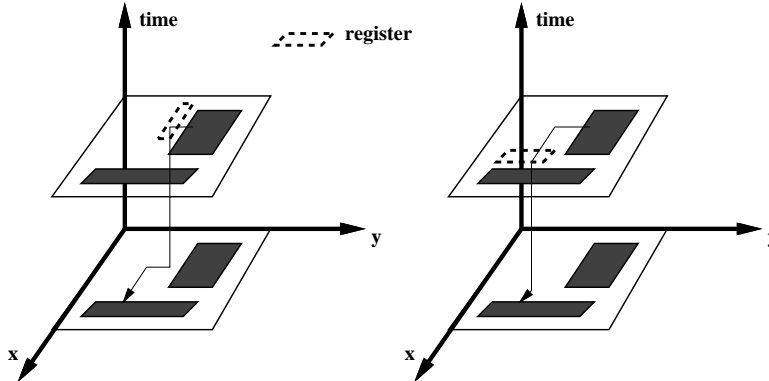


Figure 5: Temporal-Spatial routing and its relation to register allocation

develop an accurate model of hardware based on our previous work [XK961]. Next, we will use the knowledge obtained from our current research [XK962] as well as similar work to research and develop synthesis approaches so as to make full usage of the hardware model. These approaches must employ techniques which can be naturally extended to the physical domain. In fact, scheduling and allocation within high level synthesis can be looked at as placement of operations. Scheduling mainly places operations in the time domain, while allocation is a placement along a one dimensional physical model of the design. The thesis presented in [DN89] is that combined scheduling and allocation is a placement in a space-time domain. In order to accurately model the physical properties of the design hardware, the spatial component of the domain must be Euclidean (or at least Manhattan)¹.

Based on this view of the synthesis process, a new technique will be developed to attack the temporal-spatial placement problem. A typical instance of such a problem is illustrated in Figure 1. Such a technique must take full advantage of the evaluation capabilities inherent in the hardware model so as to maximally reduce the number of iterations through the partitioning procedure, therefore optimizing its efficiency. Since the problem we are facing is a placement problem, we will explore the various classes of approaches for the traditional 2-D physical placement as possible solution techniques.

An interesting feature of the temporal-spatial model is that in order to generate a full RT level design, one must perform both placement *and* routing. This is illustrated in Figure 5. A routing segment parallel to the time axis requires the existence of a register at the “via” point. Thus, temporal-spatial routing corresponds to a combined allocation of registers and their physical placement. This is an extension to the previous modeling of register allocation as a routing problem [KP87]. Clearly, this would make both the placement and routing problems inter-related and may require several iterations between the two until an acceptable solution is reached. However, the routing problem does not have to be fully solved since only the register via positions need to be determined, we will extend the RT level model described in [XK961] and use it to predict the approximate topology of the design. We will study this problem further and propose an efficient solution.

G.3 Computational Models and System-level Management

As part of this proposal we will investigate the plausibility of different models of computation that may come to fruition by utilizing programmable logic blocks as found in FPGA’s. These models

¹to be strict, the temporal domain itself must also be multi-dimensional, since conditional branches can imply that operations can share the same time-space coordinates if they are mutually exclusive.

in the past were not feasible due to lack luster speed of reconfiguring the logic blocks. But, this technology is changing rapidly, and therefore it is possible to look for adaptive systems that allow for rapid reconfiguration of the circuit due to the changes of the incoming data or algorithmic optimizations.

Based on the current technology and the projection for the future of this technology, we have identified two models of computation that are relevant to the DoD mission of design for adaptive computing facilities.

G.3.1 The Fully Reconfigurable Microprocessor Technology (FRMT): Architectural Considerations

The first model that we will study is based on the computing mission of DoD for handling existing programs that already reside on myriad of computer models and systems. These valuable code segments can be fully utilized if one to use a FPGA based model that can emulate any execution pipeline. Thus, enabling rapid development of integrated systems from a common piece of hardware for efficiency and cost reduction point of view.

As mentioned in Section D.2.3, in the FRMT, even the core CPU has reconfigurable parts. This is essential for our project since the core processor can adapt to different computation needs of the task in addition to the attached FPGA block as proposed by others. The idea of having the core processor be able to use FPGA assets to improve its performance is paramount to the notion of adaptive computing that we propose and differentiates our work from the current research in this area.

The reason this optimized core is used is that the FPGA gates are not likely to maintain the clock speeds necessary for the cutting edge microprocessor design, however, by incorporating FPGA/DPMA (Dynamically Reconfigurable Memory Array) one can "enhance" the capability of the current core. For instance, one area of enhancement is to make the decoder have a programmable subsection where different binaries belonging to different architectures are executable by decoding them into an internal format. This is similar in concept to the way pentium-pro maps X86 code into the internal micro-ops format. This capability will allow the RC to handle binaries corresponding to a selected popular architectures such as: Power PC and X86.

Another source of core enhancement is functional unit utilization. Based on the application, it is desirable to redirect the FPGA/DPMA attached block to enhance the execution phase of the operation. For instance, when vector applications are running it may be desirable to extend the DPMA to create the vector registers and other FPGA blocks to enhance the vector functional units and the related logic for exploring chaining and efficient memory accesses.

Because of the projected availability of transistors we envision that the proposed FRMT will be able to carve a niche in the self-improving microprocessor technology where a portion of the FRMT will conduct performance monitoring tasks and based on the current state and a reasonable order Markovian analysis will re-tune the core architecture to improve the performance. This ability could also be used for self-checking and reconfiguration where the on-board computer on a remote mission task is able to reconfigure the internal structure to address the faults due external sources (i.e, alpha rays etc.). An important part of the self-tuning aspect of an adaptive computer system as described in this proposal is how to tabulate performance issue parameters based on variable data inputs that are initiated by different applications. We have devised an extensive profiling technique (see Section G.3.1.3) that blends efficiently with the proposed FRMT architecture.

One important area of design is how to deal with the slow pace of reconfiguring the FPGA/DPMA as compared to the base clock of the CPU. For this problem we propose two methods that will clearly address the problem. In the first method we will utilize a careful integration of configura-

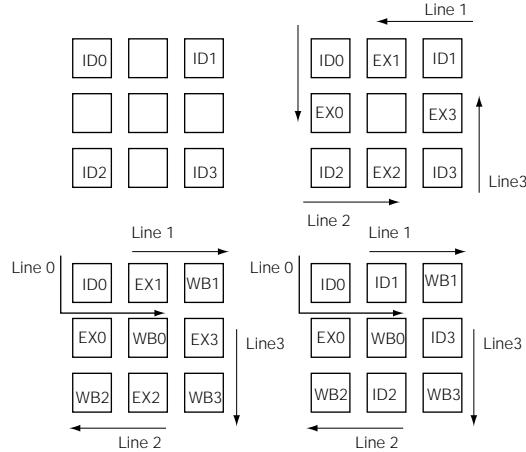


Figure 6: Superscalar Dynamic Reconfigurable Processor.

tion step of the FPGA ahead of the computation wave. This approach will resemble the interleave memory execution where the cycle time of the memory access is time multiplexed with other cycles of adjacent memory blocks. The challenge here is how to evaluate the computation delay and the area of the FPGA that is currently executing and safely reconfigure the section that is not currently "active".

The second approach is more static and allows storage of the reconfiguration bits on the chip for those tasks (e.g., in the core processor enhancement sort of blocks) that have a predefined characteristics and the data or other events will trigger the configuration to switch to the new one. This way saving the precious latency from the memory to the FRMT.

G.3.1.1 Superscalar Dynamic Reconfigurable Processor (SDRP) The inherent problem in modern superscalar architecture is the utilization of the existing resources. The architecture might have 10s of functional units, but at any given time, only a fraction of the FUs are being used. In addition to FUs, the pipeline stages itself could be wasted because of bubbles caused by branch misprediction and stalls caused by various instruction dependencies. Some of these problems have been improved upon with out-of-order issue superscalar architecture; yet, there remains to be many resources that are idle in the processor.

In a SDRP, the architecture has a number of reconfigurable elements (RE) which could be reconfigured to behave as any one of the pipeline stages or functional units on demand. This is illustrated using Figure 6. For instance, let's assume that a particular SDRP architecture has 9 REs, and initially 4 elements are decoding an instruction each (with Fetch stage sitting somewhere outside of RE blocks for this example); in this case, it is equivalent to a 4-issue superscalar. Each clock cycle it grabs one more RE as it needs either a stage or a FU to execute an instruction each clock cycle. This information about what it needs is likely to be embedded into the instruction itself; therefore, instruction needs no help from the hardware as far as knowing what it needs to complete the job. A possible problem occurs when either all the REs are being used or the routing is not available to connect to the available RE. The first problem is not really a problem in a sense that this situation is what is exactly desired, the full utilization of given resource - in this case, RE. The deadlock that may result from this could be solved easily by requiring each lines of RE (lines being a group of REs that has connected pipeline stages) to transform uniformly into next pipeline stages until more REs are available. As shown in the Figure 6, initially four REs are being used as ID stages. During the next clock cycle, each line grabs one more RE. In the 3rd clock cycle, not all lines can acquire more REs. In fact, only line 0 can grab an RE; all other lines must transform into

its following stages to execute its holding instructions in-place. Therefore, no deadlock is possible.

There are other ways of breaking deadlocks such as giving priorities to a line of REs which would preempt other lines if necessary. The routing concern is similar to the resource utilization of superscalar architecture; in SDRP, this would largely depend on the connectivity between different REs. This problem becomes almost like that of multiprocessor interconnection. The studies in this area might help, but more studies in depth are needed since in SDRP the existing line can not be crossed to pipe the current instruction stream. Despite that, if the allocation algorithm used to acquire an RE is sound, this interconnectivity conflict should be minimal. The allocation algorithms should be simple enough either to be encoded into the instruction, hardwired, or both. Another problem that has to be solved, related to the interconnectivity problem, is how the dependencies among instructions are going to be checked; this is mandatory since the consistent machine state needs to be maintained at all times. In addition, the optimal combination of a number of maximum lines and a number of REs should be decided with the profiling. For example, it would be unwise to have 4 lines given 4 REs since this would just be 4 unicycle processor executing one instruction until its completion. The relationship between the improvements made by reconfigurable models and the possible clock speed drops because of it should be studied to better measure any trade-offs.

G.3.1.2 Dynamically Programmable Memory Array (DPMA) We introduce the concept of a dynamically programmable memory array (DPMA). A DPMA uses a simple six-transistor SRAM cell for storage, and these cells are organized as arrays. A “hard-wired” processor would dedicate arrays of SRAM for a specific purpose and could not be used for any other purpose. The purpose behind the DPMA is to enhance the memory structures that exist throughout the design of a microprocessor. For instance, the DPMA may be used in critical memory cell intensive areas as: instruction and data cache, branch prediction correlation data in the form of pattern history table or branch history register, register file, and register renaming storage (i.e., mapping table and instruction window).

For example, suppose a processor has a pool of memory totaling 64 Kbytes. The 64 Kbytes could be split up into 64 individual 1 Kbyte arrays. These arrays of memory can be dynamically allocated for any purpose: instruction cache, data cache, register files, branch target buffer, and branch prediction information. An application may decide it needs more data cache than instruction cache. Later during execution, it may need more instruction cache than data cache. As another example, the branch behavior of a program may be highly predictable. The branch prediction information may be reduced, say from 2 Kbyte to 1 Kbyte, and the storage used for a more critical purpose. Alternatively, a program may be difficult to predict the branch behavior, so more storage can be allocated for branch prediction to give better dynamic branch prediction. In summary, memory arrays are allocated where they are needed the most to yield the greatest performance.

As a example consider the DPMA in Figure 7. It consists of eight memory arrays: two i-cache data banks, an i-cache tag array, d-cache data and tag arrays, a branch target buffer array, and two pattern history table banks. Using the information provided by the runtime profiler, the Architectural Optimizer notices that the data cache is performing poorly yet the branch prediction has a very high accuracy. Therefore, one of the two pattern history table banks is de-allocated and used for another data cache bank.

Using a general pool of memory to meet the needs of a reconfigurable processor presents its own set of difficulties. To begin with, a generic memory array would probably be implemented with a fixed line size. Although it is possible to implement arrays with different fixed line sizes, this increases the complexity in supporting different line sizes. Since a DPMA is going to be fully reconfigurable, it can dynamically support different virtual line sizes even when the arrays are

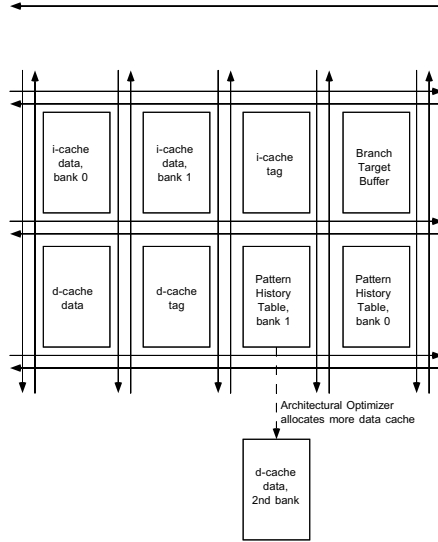


Figure 7: Architectural Optimization of the DPMA.

implemented with a fixed line size. If the virtual line size is greater than the physical line size, then multiple arrays need to be allocated at a time to simultaneously access enough bytes to meet the requirement of the physical line size. If the virtual line size is less than the physical line size and is evenly divisible, then the virtual line can be multiplexed from the physical line. If it is not evenly divisible, then two arrays need to be used at a time so that the virtual line can be self-aligned via two consecutive physical lines (similar to self-aligned instruction caches).

Since it is expected that the latency of FPGA enhanced block is longer than the optimized core, then it is perceived that the operation of the DPMA enhanced blocks will resemble the slower memory in a memory hierarchy (i.e., the relationship between L1 and L2 in the cache architecture). The design of the DPMA will be a center piece of our FPGA/DPMA cell circuit design. We will use our simulation techniques to identify the impact of DPMA on the performance given its speed differential with the core optimized section.

G.3.1.3 On-line Profiling With an architecture that can be dynamically re-configured at run-time, how can one control this reconfiguration? One way is to have a static configuration for each program which can be determined by profiling a program from previous runs. Another method is to enable the programmer to change the architectures configuration with special instructions. Finally, the architecture can be reconfigured using an “execution monitor” program, which essentially performs run-time profiling of the program and automatically adjusts the configuration parameters to boost performance. Hence, the reconfiguration of FPGAs and DPMA do not necessarily need to be programmed at compile-time, they can be dynamically re-configured based on run-time information and behavior.

An execution monitor consists of an information gatherer and a decision maker. This is illustrated in Figure 8. The information gatherer collects information about the performance of the processor. Relevant information would include stalls due to instruction cache misses, data cache misses, and data dependencies; and also penalties from mispredicted branches and instruction fetching efficiency, and the overall performance of the processor (instructions per cycle). It keeps a history of these performance results along with the configuration of the processor. The information gatherer could be implemented as a small software program that runs in parallel with the primary application. Alternatively, it could be implemented in hardware by using counters and a small

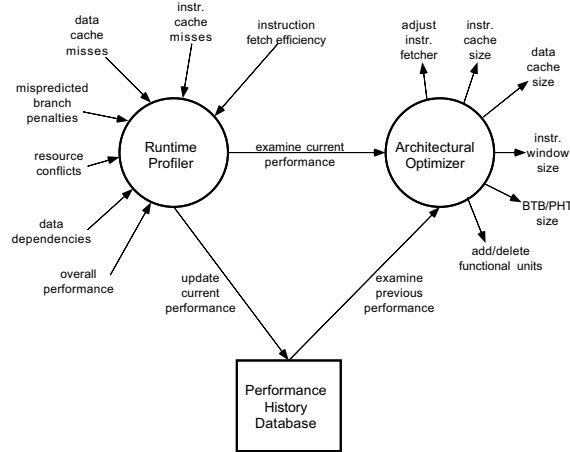


Figure 8: On-line Profiling and Architectural Optimization

memory array to record the program’s performance history.

The run-time information about a program’s performance is used by an architectural optimizer to modify the configuration of a processor’s architecture, with the goal of improving the performance. For example, if the data cache is causing too many stalls to the pipeline, the execution monitor will increase the size of the data cache by reconfiguring the DPMA. This may come at the expense of another resource, such as the instruction cache or branch prediction information. Again, the decision maker may be implemented as a software program running in parallel or as hardware. Another possibility is that the programmer could design its own custom decision maker, using FPGAs, to decide what architectural elements should be adjusted. Programs without a custom execution monitor or any profile information at all, such as legacy code, need to rely on a general purpose decision maker to adjust the processor’s resources.

A general decision maker would use several basic steps and guidelines. First, it determines which area is causing the greatest degradation in performance by examining the results of the information gatherer. Next, it decides by how much to increase the size of the target resource. Usually, this will be the smallest incremental increase allowable by the resource. The most difficult decision is to determine the victim resource, i.e., which resource(s) can be reduced. By using the history of the performance of the victim resource and the target resource, it can decide if the proposed configuration would yield a net increase in performance.

If history information is not available for either resource it may or may not decide to try the proposed configuration. It could base this decision on generic increases/decreases in performance given specific increases/decreases in a resource. If the potential decrease in performance significantly outweighs the potential improvement, then the proposed configuration may be too risky to try. After a certain period of time, the performance of this new configuration is recorded. If the performance did not improve as expected or lead to a decrease in performance, the configuration may revert back or try something else.

The more information the execution monitor gathers, the more intelligent decision it can make. For example, other information might be useful in making a decision such as the section of code currently executing or important data values. Depending on the run-time behavior of these items the execution monitor might select a configuration that ran well under similar circumstances.

To our knowledge, there has never been any research done in this area of run-time profiling. We feel this is critical to reach high performance in a reconfigurable hardware environment. With so many different possibilities of resource configurations, it will be difficult for the programmer to decide which configuration is the best at compile-time. In addition, this allows the hardware to

adapt to the run-time behavior of the program and data, something which is impossible to determine at compile-time. Furthermore, run-time profiling via the execution monitor will be important to maximizing performance from legacy code.

G.3.2 Custom Architecture Based on Dataflow Model

For the second model of computation, we will be focusing on the inherent application program parallelism available in terms of the program's data flow graph. This method has similarities to the original data flow architecture where the data flow graph representation of the program identifies dependencies and computation units that are connected together to perform a given task or tasks. The difficulty in implementing data flow architectures has been in realizing the underlying hardware to efficiently execute program data flow representations. In this context, the architectural synthesis techniques proposed in the previous section are central to this approach.

With the advent of FPGA's and in particular high-speed programmable devices the idea of direct data flow implantation becomes more viable given the flexibility of these circuits. Moreover, traditional CAM based and template organized matching stores that lead to demise of the data flow machines are no longer necessary. Thus, we propose to develop the back-end compiler tool that will generate the necessary data flow graph in terms of the reconfigurable blocks of an FPGA device. This in combination with one or more RISC microprocessors at the caliber of the ARM processor that is commonly used for embedded systems the graph is fetched from the memory and distributed to the FPGA blocks. The underlying mechanism for managing the overall operations are very similar to the standard techniques for memory and resource management commonly found in operating systems nowadays. The only difference is that memory pages can represent either hardware configuration data (or "personality modules") or computation data.

G.3.3 Overall Execution Model: Evolutionary Computation

Ultimately, a given computation can best be optimally run on a dedicated (or at least partially dedicated) processor. A straightforward approach would be to compile the computation into a control-dataflow graph and automatically synthesize a custom architecture. However, this may be a little shortsighted if done without prior analysis of the code. Analyzing the code, on the other hand, can be a tedious task and could delay the start of execution. Furthermore, code behavior can indeed depend on the input data patterns, which can vary significantly during runtime. Finally, the code may be written for a legacy processor and perhaps optimized to take advantage of that processor's special features. Thus, we propose an *evolutionary computation* model which is depicted in Figure 2.

Initially, one starts with a legacy code originally designed to run on a legacy processor. A natural approach at this point is to configure the DRA to emulate the legacy processor. Note that the emulated processor may be functionally similar to the legacy processor but can still achieve significantly higher performance because the execution pipeline is dynamically reconfigurable and can execute the code more efficiently. Another optimization can be accomplished at compile time is the introduction of a user configurable data path function as illustrated in Figure 9. By analyzing the operations performed and "loading" on-the-fly functional units personalities which may perform some of the frequently used functions much faster than the regular instructions. A typical example would be multiply-add-accumulate (or some variation) found in DSP code. In general, the machine code would simply say:

$$\text{conf } 2 \ A, B \ \rightarrow \ C$$

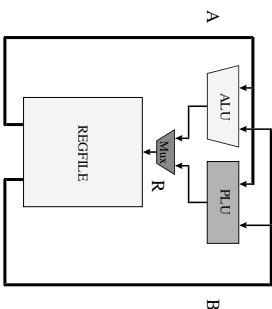


Figure 9: User-configurable Functional Unit.

instead of:

$$\text{multiply } A, B \rightarrow C.$$

In other words: **load configuration number 2 into the data path** (where conf 2 is a configuration residing in a library or on one of the context planes), **apply inputs A and B and put the result in C**. Clearly, this approach has the advantage of being code compatible with the existing code base because the existing data path could remain in place. Thus, one could change a multiplier to a coding function or to a lookup table to perform specific functions.

In addition to the functionality of the legacy processor, the emulated processor incorporates a monitoring subsystem whose task is to perform on-line profiling during code execution as described in Section G.3.1.3. The CAD tools described in Section G.2 will be used to generate the configuration of the “amended” architecture. In addition to the monitoring subsystem, another logic block (or blocks) are needed to perform some of the architectural optimization tasks as described in Section G.3.1.3. Note that it is also possible to use the core processor to execute the architectural optimization tasks through software. The final configuration of this block could be either hardware, software or a mixture of both, and will depend on the complexity and cost of the required functionality.

At the end of this phase, computation would have migrated from a legacy processor to a highly optimized processor architecture which may contain some dedicated functional blocks as well as a memory (register files, caches, etc..) structure which has been optimized specifically for the code being executed. For many applications this may represent satisfactory performance and no more optimization steps would be needed. However, for time critical applications (e.g. target recognition), real-time applications, or if more performance is required, then further improvements are needed. In this case, the computation (or some portions of it) must be run on an application-specific architecture. The CAD tools described in Section G.2 will be run to synthesize such an architecture. At this point, the profiling information already available will be used to help determining:

- the time-consuming parts of the computation,
- the memory structure best suited for the computation, and
- the special functional blocks realizing the frequently called functions.

This information can be used by the architectural synthesis tools to generate a dedicated architecture most suitable for the given computation. This architecture will either run all of the computation or may run part of it alongside the processor. If needed the monitoring and architectural optimization subsystem may still be run in order to further improve performance.

H. Related Research

The basic guidelines that define the DARPA effort in this domain has been specified in the CHAMELEON effort which sets the standard for contemporary concept of adaptive computing. One could project that future extensions of this project would lead to what we have proposed here.

As far as we can tell by evaluating the current literature, the ideas proposed here are unique and the specificity of the proposed set of tools and models of computation for the FPGA technology has not been discussed before. There are projects that are tangentially related to what we have proposed here, perhaps the closest one is the Transit project at MIT and a related project at Berkeley (BRASS). The MIT project has been going on for a number of years and has resulted in a first generation Dynamically Programmable Gate Array (DPGA) chip which has several layers of contexts thus allowing the switching from one configuration to another.

The BRASS project is just starting and proposes to integrate a processor core with a DPGA-like block on a single chip. Some early specs of the target chip are available but not the implementation.

On the other hand using FPGA's for realizing research oriented projects at the system level—not at the chip level—have been around for at least three years. At USC, an NSF funded project has been using FPGA based boards to evaluate different multiprocessor protocols for cache coherency and other bus related issues. The advantage of this scheme has been the programmability of the architecture so that the turn around time and cost is minimized in order to evaluate different contending memory interface strategies. Another project has been the Splash-2 at Stanford university where for similar reasons to the USC project a portion of this shared memory architecture is implemented in FPGA technology to allow flexible reconfiguration for optimizing the architecture.

All these projects essentially deal with statically reconfigurable models of computation. By contrast, our proposal is stemming from the fact that this programmability issue has been improved to the level that it can be done very fast compared to the processors main clock, therefore, allowing an on-the-fly reconfiguration at the machine instruction level (or in terms of the first model, data flow, at the lowest computation grain). Furthermore, our project is a more vertically integrated approach which stresses tools, methodologies and computational models. Specifically, it differs in the following aspects:

1. Logic configurations can be modified during runtime,
2. Configuration data can be modified during runtime,
3. We are proposing a Dynamically Programmable Memory Array which is specifically geared towards optimal performance of legacy code,
4. We are proposing a self-tuning architecture which can optimize the performance of legacy code based on execution history,
5. We are proposing a computational model which migrates code execution from a legacy processor to a dedicated processor in a manner completely transparent to the user, and
6. we are proposing to develop specific CAD tools to allow fast, realistic, and efficient mapping of code onto the proposed architecture.

I. Research Organization

Team Members:

- **Co-PIs:**

- *Fadi Kurdahi* received Ph.D. degree in Computer Engineering from the University of Southern California in 1987. Since 1987, he has been a faculty at the Department of Electrical & Computer Engineering at the University of California, Irvine, where he is currently an Associate Professor. His research interests are the areas of Computer-Aided Design of VLSI circuits, high-level synthesis, and design methodology of large scale systems. He has published papers and organized tutorials dealing with various aspects of chip and system synthesis at international conferences and journals.
- *Nader Bagherzadeh* received Ph.D. degree in Computer Engineering from the University of Texas at Austin in 1987. Since 1988, he has been a faculty at the Department of Electrical & Computer Engineering at the University of California, Irvine, where he is currently an Associate Professor. From 1980 to 1984 he worked for ATT Bell Labs in Holmdel New Jersey designing systems for the 5ESS telephone switching system. From 1984 to 1987 he worked for the parallel processing groups at Burroughs and MCC in Austin, Texas. His research interests are in parallel processing, advanced computer architecture, and VLSI design. He is a senior member of IEEE.
- *Robert Heaton* graduated from the University of Surrey (UK) in 1979. He has been employed in a number of positions in the silicon and networking systems industry including Texas Instruments, Phillips, Standard Microsystems, and ES2. He was the lead designer for the TI's first gate array, ARM microprocessor register file, and 100Base-T4 signaling scheme. He is currently one of the chief designers at Obsidian Technology, a silicon design consulting company in Orange County.

The research work will be done as a collaboration among all the team members. Early architectural investigation, CAD tools and user-level environment will be done at UC Irvine, and the DRPM development will be done at Obsidian Technology. The team members will meet weekly to assess the progress of the project.

The estimated efforts for F. Kurdahi and N. Bagherzadeh will be 25% for the academic year and 75% during the summer each year. In addition, three research assistants will be hired; one to work on each level of design. The RA's efforts will be 50% during the academic year and 100% during the summer. The effort at Obsidian will be 3/4 man-year each year.

J. Proposers' Previous Accomplishments

Fadi J. Kurdahi

Fadi Kurdahi has been doing research in Design Automation and VLSI design for over 13 years. Specifically, he has researched and developed CAD tools for Architectural synthesis and estimation and has published numerous papers on the subject of linking layout information to synthesis. This becomes particularly crucial for the projected fabrication technologies that are likely to implement the proposed DRA. Recently, his work has focused on FPGAs as target implementations. His work on estimation for FPGA resulted in a suite of prediction tools that are highly accurate yet runtime efficient. Architectural synthesis techniques such as scheduling and binding are being developed to utilize these estimator and generate "first-time-correct" silicon.

Since 1987, he has been a faculty at the Department of Electrical & Computer Engineering at the University of California, Irvine, where he is currently an Associate Professor. His research interests are the areas of Computer-Aided Design of VLSI circuits, high-level synthesis, and design methodology of large scale systems. His research addresses the issues of linking High Level Synthesis to subsequent levels of design, namely logic and physical levels. He has published papers dealing with various aspects of chip and system synthesis at international conferences and journals. He has organized and participated in tutorials on high level synthesis at international conferences such as the Design Automation Conference in 1990. Prof. Kurdahi was an Associate Editor for IEEE Transactions on Circuits and Systems II in 1993-1995, and was a guest editor for a special issue of the VLSI Design Journal on "Linking Behavioral, Structural and Physical Models of Hardware". He was Organization chairman of the Sixth International High Level Synthesis Workshop, and served on the Program committees of several conferences and workshops such as: the International High Level Synthesis Workshop, the International Symposium on System Level Synthesis, the ACM/IEEE Physical Design Workshop, and the European Design and Test Conference. He received the Research Initiation Award from the National Science Foundation in 1989, and the ACM/SIGDA Fellowship in 1991 and 1992. Prof. Kurdahi is a member of IEEE and ACM.

Nader Bagherzadeh

Nader Bagherzadeh has been involved in the design of advanced microprocessor architectures for the past ten years. He was leading designer of the first 4-issue VLIW microprocessor called VIPER. The VIPER architecture demonstrated pioneering work in the areas of processor optimization for routing operand data through bypass circuitry. Moreover, it utilized a unique pipeline architecture with a refined addressing mode that mitigated the development of future VLIW processors by industry. Later on he worked on an out-of-order issue superscalar microprocessor called SDSP targeted for multimedia computation intensive applications. This study lead to several new discoveries in the areas of instruction fetching and register renaming hardware optimization.

Currently he has been combining his experience with out-of-order issue superscalar with multithreaded architectures. In this work he is the first researcher that demonstrated the design of a register renaming scheme based on the reorder buffer and instruction window that can handle multiple threads. This work will open the way for exploiting instruction level parallelism beyond the wide issue superscalar model. Finally, he is involved in the hardware design of a 500 Mhz reorder/buffer that utilizes the True Single Phase clocking scheme.

Robert Heaton

Has been designing VLSI chips for over 15 years. He was the lead designer of TI's first Gate Array. In addition, he was responsible for the VLSI design group at Acorn computers where he developed the ARM processor 3 chip set: CPU, memory controller, and display controller. As the first commercially available RISC processor the ARM has been one of the most successful embedded controllers in the market. It has recently been endorsed by Microsoft as one of only two processors to be qualified for the CE operating system.

As senior director of engineering at Standard Microsystems Corp., he lead a number of design teams that implemented several successful mixed signal networking chips which were responsible for some \$650M worth of product sales during the early nineties. He holds a number of patents related to various aspects of chip design and communication protocols including the fundamental patent for 100BaseT4, the first 100Mb/S LAN signaling scheme over voice grade cable standardized by the IEEE.

During 1996 Robert Co Founded Obsidian Technology CORP which undertakes contract research and development in the VLSI computing and LAN fields

K. Description of Facilities

The available facilities at UCI include a large number of UNIX and PC workstations supported by several high-powered SUN servers. These computers are networked via a campus-wide FDDI backbone. In addition, the PIs have access to a wide variety of CAD tools for both hardware and software design. Such tools include: Mentor Graphics, Compass, Viewlogic, Vantage, Cadence, Synopsys and Xilinx. A fully-equipped FPGA design lab is available for usage to prototype hardware using the Xilinx XC4000 FPGA evaluation boards. This is supported by a suite of the latest Xilinx XACT tools running on both PCs and SUNs. Campus-wide computing resources include a CONVEX supercomputer and access to Crays in the San Diego Supercomputer center.

L. Cost Breakdown

SUMMARY

Salaries & Wages	\$298,958	
Employee Benefits	\$17,938	
Total Salaries, Wages & Fringe Benefits		\$316,896
Permanent Equipment		\$40,000
Participant Support Costs		\$0
Travel		
Domestic	\$30,000	
Foreign	\$0	
Total Travel		\$30,000
Other Direct Costs		
Materials	\$19,000	
Publication Costs/Page Charges	\$0	
Consultant Services	\$0	
CHIP Fabrication (MOSIS)	\$100,000	
Subcontracts	\$300,000	
Other-(Tuition & Fees)	\$131,314	
Total Other Direct Costs		\$550,314
TOTAL DIRECT COSTS		\$937,210
INDIRECT COSTS (Total Direct Costs less Equip,T&F @ 49.9%)		\$244,957
TOTAL DIRECT & INDIRECT COSTS		\$1,182,167

Year 1 07/01/97 to 06/31/98

Name	Title	Emp. Type	C/Inr Year	No. Emp.	Monthly			Total Salary	Bene. % Rate	Employ. Benefits	TOTAL	
					Rate	% Time	Type No.					
P.I.	Assoc.III	Fac:Sun	1997	2	7,656	100%	S	2	\$30,624	9.60%	\$2,940	\$33,564
	GSR 3	Student	1997	3	2,716	50%	A	9	\$36,666	1.80%	\$660	\$37,326
	GSR 3	Student	1997	3	2,716	100%	S	3	\$24,444	3.40%	\$831	\$25,275
	Adm.Asst.	Staff	1997	1	2,582	10%	C	12	\$3,098	25.00%	\$775	\$3,873
TOTAL									\$94,832		\$5,206	\$100,038
PERMANENT EQUIPME (Two SUN Ultras Workstations)										\$20,000	\$20,000	
TRAVEL:												
DOMESTIC: (Three PI trips to ARPA+conferences, three conferences for students)										\$10,000		
FOREIGN:											\$10,000	
PARTICIPANT SUPPORT COSTS:											\$0	
OTHER DIRECT COSTS:												
MATERIAL & SUPPLIES:										\$6,000		
PRINTING, REPRODUCTION, REPRINTS:												
CONSULTANT SERVICES:												
COMPUTER SERVICES:												
SUBCONTRACTS:										\$111,000		
OTHER: Tuition & Fees										\$39,672		
Tuition & Fees (three non-resident students)										\$39,672		
Total Other Direct Costs											\$156,672	
TOTAL DIRECT COSTS											\$286,710	
INDIRECT COSTS: Total Direct Costs less Equip, Tuition & Fees =										\$141,038	at 49.90%	\$70,378
TOTAL DIRECT AND INDIRECT COSTS											\$357,088	

Year 2 07/01/98 to 06/31/99

Monthly												
Name	Title	Employe Type	Clnr	No.	Rate	% Time	Type	No.	Total Salary	Benefit % Rate	Employ. Bene.	TOTAL
P.I.	Assoc.III	Fac:Sun	1998	2	8,039	100%	S	2	\$32,155	10.10%	\$3,248	\$35,403
	GSR 3	Student	1998	3	2,852	50%	A	9	\$38,499	2.30%	\$885	\$39,384
	GSR 3	Student	1998	3	2,852	100%	S	3	\$25,666	3.90%	\$1,001	\$26,667
	Adm.Asst.	Staff	1998	1	2,711	10%	C	12	\$3,253	25.50%	\$830	\$4,083
TOTAL									\$99,573		\$5,964	\$105,537
PERMANENT EQUIPME (Two SUN Ultras Workstations)											\$20,000	\$20,000
TRAVEL:												
DOMESTIC: (Three PI trips to ARPA+conferences, three conferences for students)											\$10,000	
FOREIGN:												\$10,000
PARTICIPANT SUPPORT COSTS:												\$0
OTHER DIRECT COSTS:												
MATERIALS AND SUPPLIES:												\$6,000
PRINTING, REPRODUCTION,REPRINT:												
CONSULTANT SERVICES:												
COMPUTER SERVICES & MAINTENANCE:												
SUBCONTRACTS:												\$90,500
OTHER:												\$43,639
Tuition & Fees (three non-resident students)											\$43,639	
Total Other Direct Costs												\$140,139
TOTAL DIRECT COSTS												\$275,676
INDIRECT COSTS: Total Direct Costs less Equip, Tuition & Fees									\$121,537	at	49.90%	\$60,647
TOTAL DIRECT AND INDIRECT COSTS												\$336,323

Year 3 07/01/99 to 06/31/00

Monthly												
Name	Title	Emp. Type	Clnr No.	Year	Rate	% Time	Type	No.	Total Salary	Benefit % Rate	Employee Benefits	TOTAL
P.I.	Assoc.III	Fac:Sun	1999	2	8,441	100%	S	2	\$33,763	10.60%	\$3,579	\$37,342
	GSR 3	Student	1999	3	2,994	50%	A	9	\$40,424	2.80%	\$1,132	\$41,556
	GSR 3	Student	1999	3	2,994	100%	S	3	\$26,950	4.40%	\$1,186	\$28,136
	Secretarial Staff		1999	1	2,847	10%	C	12	\$3,416	25.50%	\$871	\$4,287
TOTAL									\$104,553		\$6,768	\$111,321
PERMANENT EQUIPMENT:											\$0	
TRAVEL:												
DOMESTIC: (Three PI trips to ARPA+conferences, three conferences for students)											\$10,000	
FOREIGN:											\$10,000	
PARTICIPANT SUPPORT COSTS:											\$0	
OTHER DIRECT COSTS:												
MATERIALS AND SUPPLIES:											\$7,000	
PRINTING, REPRODUCTION, REPRINTS:												
CONSULTANT SERVICES:												
CHIP FABRICATION: (MOSIS)											\$100,000	
SUBCONTRACTS:											\$98,500	
OTHER:											\$48,003	
Tuition & Fees (one non-resident student)											\$48,003	
Total Other Direct Costs											\$253,503	
TOTAL DIRECT COSTS									\$228,321	at	49.90%	\$374,824
INDIRECT COSTS									\$228,321	at	49.90%	\$113,932
TOTAL DIRECT AND INDIRECT COSTS												\$488,756

OBSIDIAN 3 Year Budget

Overview

The main expense element is personnel costs of about half of which is for two graduate students on a part time basis and half is allocated to experienced engineering staff. Capital expenses are largely for simulation software and personal computer equipment.

Breakdown of Cost Estimates

	<i>Year 1</i>	<i>Year 2</i>	<i>Year 3</i>	Total
Personnel	80,000	80,000	89,000	249,000
Equipment	20,000	2,500	2,500	25,000
Travel	4,000	2,000	1,000	7,000
M&S	2,000	2,000	2,000	6,000
Other	5,000	4,000	4,000	13,000
Year Total	111,000	90,500	98,500	\$300,000

Personnel

6250 Man Hours @ \$20/hour	\$125,000
1771 Man Hours @ \$70/hour	\$124,000
TOTAL	\$249,000

Equipment Purchase & Rental

PC's	\$6,000
Logic simulation software	\$10,000
Circuit simulation software	\$5,000
Lab hardware	\$4,000
TOTAL	\$25,000

M&S

Copying	\$3,000
Telephone charges	\$3,000
TOTAL	\$6,000

Travel

Travel: 1 Trip to Arpa conference + 1 conference/year	\$7,000
---	---------

Section III. Additional Information

References

- [MIT] MIT Transit Project. http://www.ai.mit.edu/projects/transit/rc_home_page.html
- [BRASS] Berkeley BRASS Project. <http://HTTP.CS.Berkeley.EDU/projects/brass/>
- [BYU] FPGAs, reconfigurable logic: related publications. <http://splash.ee.byu.edu/docs/papers1.html>
- [BYU2] SPLASH 2 : FPGAs in a Custom Computing Machine. <http://www.computer.org/cspress/catalog/bp07413.htm>
- [BYU3] D.A. Clark, *Supporting FPGA microprocessor through retargetable software tools*. Master's Thesis, Brigham Young University Electrical and Computer Engineering Department, 1995.
- [PH95] R. Petersen, B.L. Hutchings, *An Assessment of the Suitability of FPGA-based systems for use in Digital Signal Processing*. 5th International Workshop on Field Programmable Logic and Applications, pp293-302, August 1995, Oxford, England.
- [DEC] SA-110, StrongARM microprocessors, Digital Equipment Corp. <http://www.digital.com/info/semiconductor/sa110.htm>
- [XK961] M. Xu and F. Kurdahi, *Area and timing Estimation of LUT-based FPGAs for High-level Applications*. Proc. European Design and Test Conference, 1996.
- [XK962] M. Xu and F. Kurdahi, *Layout-Driven synthesis techniques for High-Level Synthesis*. Proc. 9th Intl. Symposium on System Synthesis, 1996.
- [KP87] F. J. Kurdahi and A. C. Parker, *REAL: A Program for REGISTER ALlocation*. Proc. Design Automation Conf., 1987.
- [RK92] D. S. Rao and F. J. Kurdahi, *Partitioning by Regularity Extraction*. Proc. Design Automation Conf., 1992.
- [DN89] S. Devadas and A. R. Newton, "Algorithms for hardware allocation in data path synthesis," *IEEE Trans. CAD*, vol. 8, no. 7, pp. 782-794, 1989.
- [XLNX] Xilinx 6200 Databook. <http://www.xilinx.com/partinfo/6200.pdf>

OBSIDIAN TECHNOLOGY

37 Niguel Pointe Drive,
Laguna Niguel
CA 92677.
(714) 363-7982. Fax 249-2167.

7 February, 1997

Professor Fadi J. Kurdahi,
ECE Department
University of California
Irvine, CA 92697

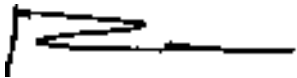
Dear Fadi

This letter expresses our commitment to participate in the proposal to DARPA entitled: **Dynamically Reprogrammable Architecture: A new Approach to Adaptive Computing**, which is submitted in response to the DARPA BAA97-06.

As part of this project, Obsidian will be responsible for the low level design and simulation of the DRA chip. Further details on our involvement are discussed in the proposal.

I am looking forward to a successful proposal and a fruitful cooperation between UCI and Obsidian.

Sincerely,



Robert Heaton
Obsidian Technology

heaton@deltanet.com

Copies of Relevant Papers