

# Design Methodology of a Low-Energy Reconfigurable Single-Chip DSP System

Marlene Wan, Hui Zhang, Varghese George  
Martin Benes, Arthur Abnous, Vandana Prabhu  
Jan Rabaey

Electrical Engineering and Computer Sciences, University of California at Berkeley  
Berkeley, CA. USA

**ABSTRACT** - In this paper, we first present a reconfigurable architecture template for low-power digital signal processing, and then an energy conscious design methodology to bridge the algorithm to architecture gap. The energy efficiency of such an architecture and the effectiveness of the methodology are demonstrated in case study implementations targeting baseband voice processing and digital signal processing.

## 1. INTRODUCTION

The future of portable devices demand an increasing support of multiple standards of communication, audio and video CODEC algorithms. This flexibility requirement points to the need for programmable devices. In addition, the need for low-power design will continue to be predominant in wireless devices. The combined requirement on providing flexibility, low-power and high-performance is a challenging design issue for future digital systems. As shown in Fig. 1, while ASICs can achieve the lowest power consumption, the flexibility requirement of the future systems cannot be met by building dedicated systems. Residing at the other end of the flexibility spectrum, traditional instruction-based micro-processor or DSPs either fall short in terms of performance or are too power inefficient. Previous research on reconfigurable computing for DSP applications [1] has shown performance superiority when compared to instruction-based programmable devices at the expense of sacrificing area or power.

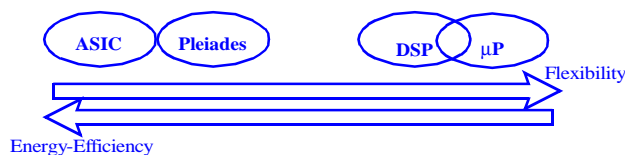


Figure 1. Energy and Flexibility Spectrum for Different Architectures

It is only until recently that reconfigurable systems are considered as a candidate to deliver lower-power solutions[2][3]. A low power reconfigurable DSP architecture template (Pleiades[3]) which encapsulates heterogeneous computing elements has been proposed [4][5] to solve the problem of meeting the requirement of flexibility, speed and energy efficiency at the same time. The Pleiades architecture

style echoes the current trend in system-on-a-chip design which includes a wide variety of macromodules including core processors, DSPs, programmable logic, embedded memory, and custom modules [6]. One of the big challenges in building such a system is to create an architectural exploration environment that provides the designer with meaningful design guidance and a means to evaluate different choices and their impact in the different phases of a design. One of the main requirements of such an environment is that it can effectively deal with the multiple levels of programming granularity [7] (hardware-software-configurable code-sign). Of the limited design environments [8][9] which address the issues raised above, a majority of them focus almost entirely on the timing and area, with only casual references to energy or power. As power dissipation represents one of the major constraints in today's embedded systems, a meaningful exploration environment should consider energy, delay, and area on the same footing.

In this paper, we describe in detail one such energy-conscious design methodology to address the design exploration problem for heterogeneous reconfigurable systems. We will first introduce the Pleiades architecture template (Section 2). The design methodology is discussed in Section 3 and results for several architecture implementations using the methodology is presented in the final section. Since analysis of both area and performance cost functions is rather well understood, the bulk of our attention in this paper will be devoted to the power models and predictors.

## 2. HETEROGENEOUS RECONFIGURABLE DSP

Reconfigurable architectures [10][11][12] have received significant attention in recent years in both the general purpose computing as well as the embedded processing. Mixing processor with fine-grain reconfigurable elements has been the main approach attempted by the above systems. The Ple-

pleiades reconfigurable architecture achieves low energy consumption by providing a computational platform with mixed programming granularity (i.e. microprocessor, reconfigurable dataflow, FPGA). In this section, we explain our architecture concept, and then provide a description of the reconfiguration and computation models used in our design methodology.

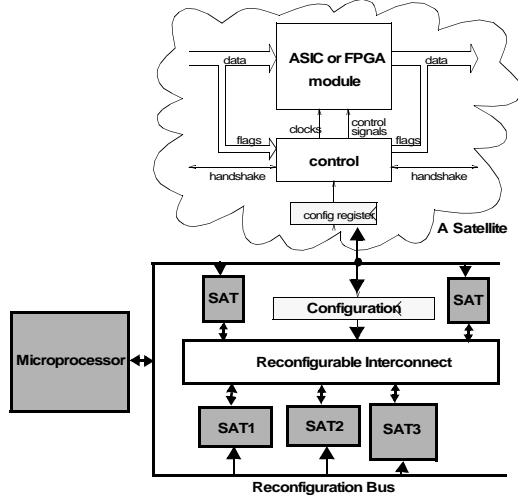


Figure 2. Heterogeneous Architecture Template

## 2.1 Architecture Template

The Pleiades architecture (Fig. 2) is composed of a programmable microprocessor and heterogeneous computing elements (referred to as satellites in the rest of the paper). The architecture template fixes the communication primitives between the microprocessor and satellites and between each satellite. For each algorithm domain (communication, speech coding, video coding), an architecture instance can be created (with known satellite types and numbers)

To reduce overhead in terms of instruction fetch and global control, the architecture utilizes distributed control and configuration. To achieve distributed control, each satellite is equipped with an interface that enables it to exchange data streams with other satellites efficiently, without the help of a global controller. The communication mechanism between each satellite is dataflow driven. The control means available to the programmer are basic satellite configurations to specify the kind of operation to be performed by the satellite, and configurations for the reconfigurable interconnect to build a cluster of satellites. All configuration registers are part of the processor’s memory map and configuration codes are memory writes from the processor’s point of view.

## 2.2 Model of Computation and Reconfigura-

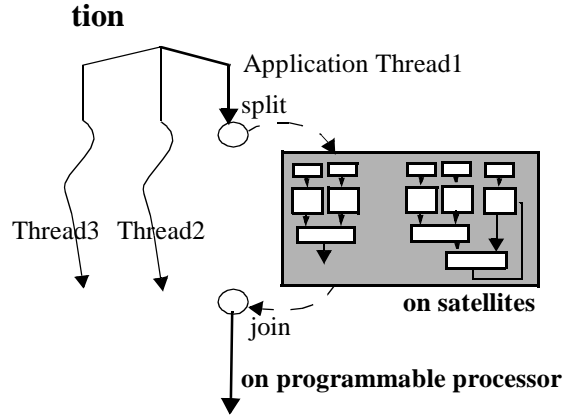


Figure 3. Flow of Computation on Pleiades

While multiple threads of application can be run on an instance of the Pleiades architecture template, the compilation of a single thread down to the reconfigurable components is the main core of the higher level scheduling tools that can utilize multi-threads. Therefore, the methodology described in the rest of the paper aims to support a smooth transition from a single thread algorithm to an optimized implementation on Pleiades. Fig. 3 illustrates the flow of computation supported by this software methodology. As shown in the figure, a sequential thread is first initialized on the microprocessor. After configuration codes are executed on the processor, the control is transferred to the Pleiades reconfigurable satellites and the computation is returned back to the processor after all satellite operations are finished.

The main idea behind reconfigurable computing that is advocated by the Pleiades system is to build a computational engine through spatially-programmed connections of processing elements (satellites). The interconnect model that needs to support such a system is depicted in Fig. 4. Each bar on the time-axis represents a set of inter-satellite connections that has to be realized simultaneously. Note that a configuration period can vary from a single duration-limited function to the entire duration of an application in a multi-function system [13].

## 3. DESIGN METHODOLOGY

### 3.1 Overview of the Methodology

There are two key issues to be resolved in order to make the methodology practical to the designers. Firstly, the architecture combines two very distinct models of computation, control-driven computation on the general-purpose microprocessor and data-driven computing on the clusters of satellites. Therefore, the goal of the architectural exploration process is to partition the application over these two

Architecture Instance:  
3 Address Generators, 3 Memories, 1 MAC/MUL and 1 ALU

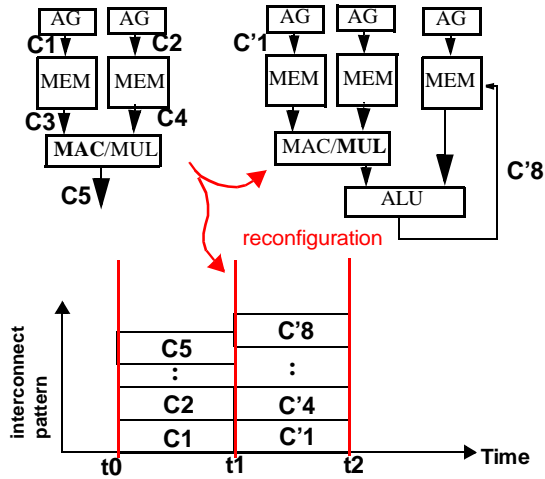


Figure 4. Model of Reconfiguration

computing paradigms so that performance and energy dissipation constraints are met (during the compilation process). Secondly, optimizations related to reconfigurability have to be supported at both the architecture design as well as compilation stage. Both of these issues requires careful modeling of the algorithm and the underlying heterogeneous architectures.

The basic flow of the design exploration methodology [14] is presented in Fig. 5. After the introduction of terminology and a short overview of the overall flow (3.1.1), a detailed description of each of the components is given in section 3.2 to section 3.5.

Definition of **Kernel** - A computational intensive part of the algorithm that often resides in nested loops.

Definition of **Macromodel** - A quantified view of the cost functions of the design module in terms of its parameters and constraints.

A macromodel can take on many forms: it can be represented by a fixed number, a table, a set of equations, an invocation of a tool such as an estimator, or a composition of a set of other macromodels. The main benefit of macromodels is that they may be used as a black box so that a designer interacts with designs in the same manner regardless of the subcomponents. For example, by using macromodels, the effects of changing a hardware library can be instantly evaluated without rewriting the behavioral description, allowing the designers to concentrate more on exploration and optimization.

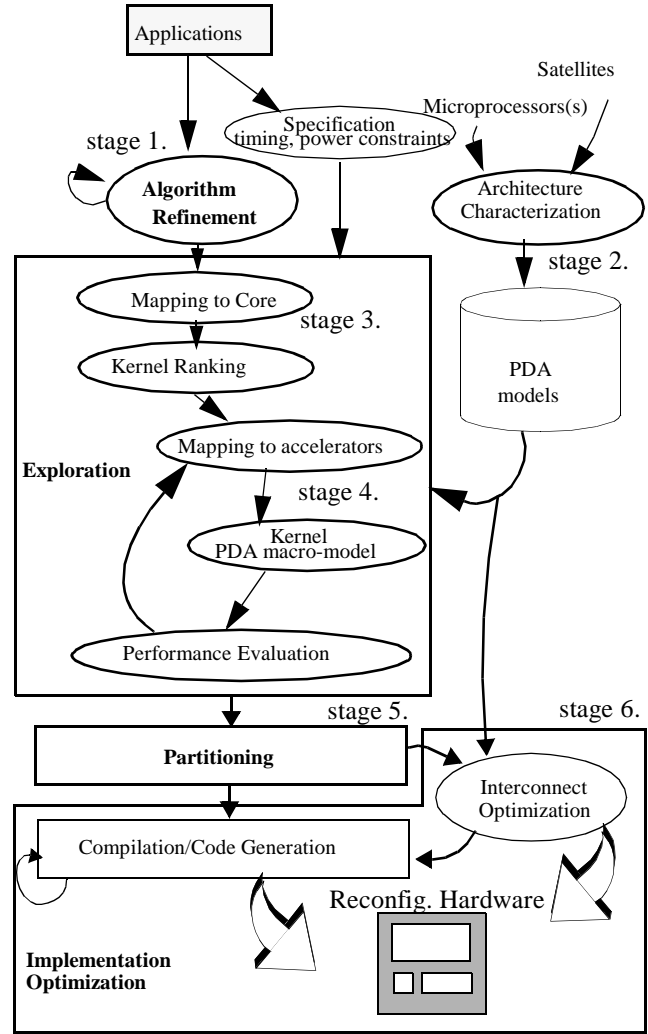


Figure 5. Software Methodology Flow

### 3.1.1 Basic Methodology Flow

The methodology flow takes DSP or communication algorithms specified in a high-level language (e.g. C) as input. The initiation of the design process requires the establishment of a first-order baseline model of the algorithm complexity and bottlenecks. Such a model allows for the selection and execution of architecture-independent optimizations (stage 1). As architectural choices have yet to be made, this model assumes the presence of a “virtual architecture” with some generic operator costs attached to it. Optimizations at this stage only address either win-only situations or order-of-magnitude improvements, so that absolute accuracy is not that important.

After a satisfactory algorithm formulation is obtained, the architectural mapping and partitioning process can be entered. To be meaningful, the partitioning process should be based on realistic bottom-up information regarding the

cost of implementing functions and operations on the different architectural choices. Our design-exploration methodology relies extensively on the availability of Power-Delay macromodels for all components in its architectural library (stage 2). The estimation methods employed in each of these models vary depending upon the type of the module and the desired accuracy. While the absolute accuracy of these characterizations is not crucial, it is important that bounds on the prediction accuracy are known. “Improvements” that fall within the noise level of the estimations should be treated warily.

The architecture partitioning and mapping process is started by establishing an initial solution. Given the implementation simplicity of a pure software implementation, we have adopted a “software-centric” approach that assumes that the whole algorithm is initially mapped onto the core processor (stage 3). This establishes how close such a solution adheres to the design specifications and helps to establish the design bottlenecks. A rank ordering of the dominant compute kernels is established. Dominant kernels are evaluated in order of importance. If a hardware implementation is deemed worthwhile, a repartitioning of the design is established (stage 4 and stage 5).

After all costly kernels are mapped to accelerators, a final partition of the algorithm across different architectures is obtained. While the rest of the algorithm remains as high-level language, the portions of the algorithm to be implemented by satellites are specified in an intermediate form that is capable of modeling the structure of the reconfigurable satellite operations (i.e. as a netlist). Based on this conceptual netlist, implementation optimizations (stage 6) are invoked to choose a good reconfigurable interconnect architecture (during architecture design stage) and to generate efficient configuration and interface code (during compilation stage).

At different phases of the design phase, it is important to show the impact of particular design choices on the overall performance and energy of the application in order to give meaningful design guidance. A spreadsheet-like environment proposed in [15] does precisely that and it is utilized in our methodology.

### 3.2 Algorithm Characterization and Bottleneck Detection

Modern DSP algorithms such as voice and video CODECs combine regular but computation-intensive kernels (typically the dominant factor) with irregular control functions. While the latter is best implemented on a traditional processor architecture, the former presents ample opportunity for design optimization by exploiting hardware accelerators.

It is important to identify potentially power-hungry portions of an algorithm as early as possible. The *inherent com-*

*putational complexity* (counts of basic operations and memory accesses) is a meaningful measure to identify dominant kernels [16]. This information can be obtained through a combination of dynamic profiling (loops and conditionals) and static analysis (within basic blocks) of the high-level specification. Since operations do not have a uniform cost, a weighted sum (where the weights indicate the energy allocated to each operator in the abstract implementation model) is calculated and aggregated at either the function or basic block level to indicate the power consumption trend within the application.

Based on this information, the designer can rewrite the code to either reduce the inherent cost of the algorithm, or extract kernels into procedure calls. These procedure calls are then considered for potential hardware acceleration.

The algorithmic refinement and kernel extraction process is based on a combination of commonly available tools. The algorithm is simulated with appropriate input vectors (representing standard operation), and profiling information is gathered at the basic block level (e.g. using the `gcc -a -g` options combined with post-processing). The profiling frequency is back-annotated to the source code to provide a first-order complexity-breakdown. Combining the basic-block execution frequencies of the profiler with a breakdown of the blocks into basic operations, as provided by standard compiler parser front-end [17], generates a detailed and illustrative overview of the distribution of the algorithm complexity over basic operators and memory accesses.

### 3.3 Architecture Characterizations and Modeling

A successful system modeling technique relies not only on the use of macromodeling, but also on a careful characterization of the underlying architectural choices. We introduce several modeling methods for the programmable devices in the target architecture and discuss how macromodels can be built using these primitives.

Instruction-level modeling has emerged as the preferred method for characterizing the energy consumption of general-purpose cores [18]. The energy base cost of each instruction is obtained by either physical measurement (e.g. [18]), physical or RTL-level simulation, or vendor supplied technology notes [19]. Each instruction base-cost includes a scaling factor introducing the effects of frequency and voltage. Given a piece of assembly code, the total energy consumed is the sum of the base costs of the executed instructions. More accuracy can be obtained by adding inter-instruction effects, as well as correction factors for pipeline stalls and cache misses.

Currently, we have evaluated the instruction base-costs and pipeline stall effects for several ARM cores [20] through physical measurements performed on ARM evaluation boards as well as physical simulation in PowerMill. In

addition, a profiler (included in ARM SDK2.1) is modified to record the energy at each instruction boundary to trace the dynamic behavior of the algorithm.

The power-characterization of the satellite modules is somewhat more complex due to the variability of the units, difference in levels of programmability and influence of factors such as signal correlations. However, moving components from soft- to hardware are only worthwhile when resulting in substantial improvements, making some inaccuracy acceptable. Furthermore, adequate macro-modeling techniques have been developed for parameterizable functional modules such as memories, multipliers, ALUs, etc. [21]. Analytical models of the effective switching capacitance  $C_{eff}$  are derived from circuit-level simulations, hence including effects such as short-circuit currents. Given the unknown nature of the signal statistics in a given accelerator configuration, a (pessimistic) white-noise signal distribution is assumed during characterization. The total energy spent by a given functional module is then modeled as:

$$Energy_{FU} = C_{eff} \times N \times V_{dd}^2 \quad (1)$$

where  $N$  equals the number of accesses of the module. The units for  $C_{eff}$  and  $V_{dd}$  are  $pF$  and  $V$  respectively.

For fine-grained programmable units such as embedded FPGAs, accurate energy estimations can only be obtained after mapping, placement and routing are performed. Currently, all of the functions we are implementing on FPGAs are small datapath elements (array of adder, comparators etc.). Therefore, manual placement and routing are performed and energy collected from simulating the functions on a low energy FPGA [22] using Powermill. In general, for larger functions, it is advisable to develop characterized libraries of macro-modules to avoid time-consuming mapping and analysis steps to be performed during high level exploration. One way to use such macro-modules is to perform fast placement and mapping at the algorithmic level [23].

The reconfigurable interconnect module between satellite processors can consume considerable energy. Models for estimating interconnect power have been proposed at the algorithmic level [24]. Given the size and number of the functional modules in the accelerator network, it is possible to predict the average length and capacitance of each wire. The total energy can then be estimated using the following expression:

$$Energy_{INTER} = \left( \sum_{i \in network} C_i \times V_{dd}^2 \right) \times N_{inter} \quad (2)$$

with  $N_{inter}$ , the number of accesses to the network. The units for  $C_{eff}$  and  $V_{dd}$  are  $pF$  and  $V$  respectively.

While the above interconnect model is very suitable for algorithmic level estimation and optimization, it is not adequate for architecture level optimizations. Therefore, a more sophisticated interconnect model is developed based on the module floorplan and reconfigurable interconnect architec-

ture. This model is carefully documented in [25].

## 3.4 Exploration and Partitioning

### 3.4.1 Base cost

Given the desirability of a software implementation, it is a natural choice to first explore the power and timing performance of the application when implemented entirely on the processor core. This can be obtained by combining the refined algorithm obtained from stage1 and the microprocessor energy (timing) estimation tool in stage 2 (Fig. 5). These costs establish a baseline to measure the impact of architectural optimizations. It can also be used to evaluate compliance with timing and energy constraints and the range of improvement that has to be obtained.

Because of the inefficiency of most general purpose cores, some of these constraints may not be satisfied. To identify areas of substantial improvement, a relative ranking of the dominant kernels is established. Using the dynamic instruction-level code profiler, a percentage breakdown of the energy consumed by each function (and basic block) is extracted and presented in a hierarchical call-tree format (Fig. 6). Kernels can only reside at leaf nodes and its energy and performance models can be changed to represent different implementation choices.

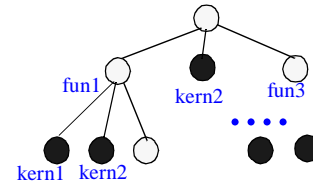


Figure 6. Conceptual representation of the design

### 3.4.2 Kernel Macromodels

The most costly kernels selected in stage 3 are mapped to the Pleiades satellites. The power performance of the kernels are re-evaluated by building a macromodel of the kernel and using information obtained from the architecture characterizations. For estimation purposes, a number of mapping strategies can be pursued:

- Simplified mapping taking only a couple of algorithmic properties into account (such as operation and memory access counts).
- “Relaxed” or fast mapping delivering well-established bounds on speed and energy for a given kernel. These techniques were developed and have been used effectively by the high-level synthesis community [26].
- Libraries of commonly-used kernel mappings.
- Manual mapping.

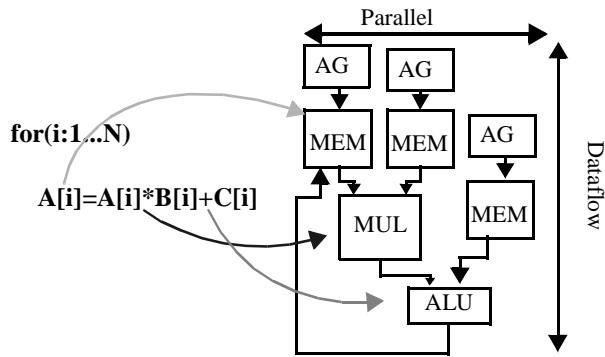


Figure 7. Direct Mapping of a Kernel to Pleiades

Our current approach uses a mix of the above. The Pleiades satellite computations are actually very close to the structure of communication and DSP algorithm kernels. A direct mapping (shown in Fig. 7) of the algorithm often gives a good mapping on Pleiades, making a manual exploration approach a practical and even attractive choice.

For all kernel mappings, it is important to have an unified representation for the mapped structures. Since each kernel is mapped to clusters of satellites, an object-oriented intermediate form based on the concept of modules (heterogeneous satellites) and queues (links between satellites) is created. A mapped kernel is constructed by building a netlist using the module and queue library (Fig. 8). In order to facilitate performance feedback in the manual mapping process, wrappers are also placed around all modules and queues so modules can be modeled as concurrent processes and queue as synchronized objects. Energy and time stamps are also associated with each modules and queues so performance can be collected. A kernel specific simulator as its macromodel is automatically instantiated once a netlist is specified.

Currently, the intermediate form is implemented in the C++ language and the Solaris thread library [27] (other common thread libraries can be switched in easily). Common satellite processors (such as MAC/multiply processor,

ALU processor, memory and address generator etc.) have been incorporated in our library.

### 3.4.3 Hardware-Software Partitioning

The ranking obtained in stage 3 establishes a ground rule for the amount of possible improvement that can be made in the style of Amdahl's law. The resulting ordering obtained in this stage is used to guide the ensuing optimization strategy. In this context, when the optimization function is convex, local optimizations is considered to be global for both speed and energy, i.e. optimization of a single kernel directly translates into a global optimum. We move kernels one by one to the Pleiades satellite according to the ordering until no more kernels can be found or a satisfactory performance has been obtained. The ordering in performance is used first when the original implementation does not meet the timing constraint. After the timing deadline is met, the ordering in energy is used.

For the case when the percentage of kernel performance (or energy) in software implementation does not indicate proportional saving in new implementations, optimal partitioning can be achieved by formulating the optimization process as an ILP problem as described in [28] and solved using heuristics.

## 3.5 Implementation Optimizations

While reconfigurability provides us with the flexibility required by the applications, high energy consumption of interconnects in related implementations (e.g. FPGA architectures [29]) and large configuration cycle time in such systems [30] reminds us that careful optimizations are needed for both reconfigurable interconnect architecture and configuration codes. Using the output of the partitioning stage (a list of netlists corresponding to mapped kernels running on Pleiades satellites), a straight forward yet effective method is used to first create an architecture instance (decision on the number of modules needed). The focus of our design environment is to provide evaluation and optimization methods to assist the design explorations of interconnect architecture and configuration code generation.

### 3.5.1 Architecture Instantiation

Given all the kernel netlists to be run on the final architecture, the number of each computational satellite (MAC, ALU, FPGA, etc.), defined as *CompSat*, is set to be the maximum number of *CompSat* among all kernel netlists.

The number of memory satellites is more difficult to determine. Consider the example where kernel 1 requires concurrent data access from array A and B, kernel 2 from B and C, kernel 3 from C and A. The maximum number of memories among all kernel is 2. However, in order to satisfy the required concurrency, the optimal number of memory is

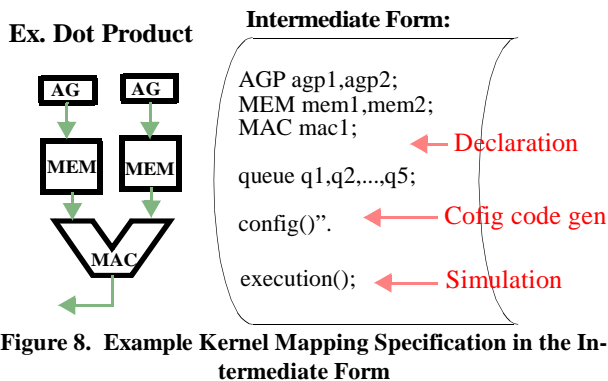


Figure 8. Example Kernel Mapping Specification in the Intermediate Form

3. On the other hand, it is often not realistic to assign a memory satellite for each array in the algorithm (because it will result in large number of small memories of variable sizes). The methodology allows the designers to evaluate several memory configurations of their choice (a memory configuration provides the number of memories and size of each memory) and decide on a configuration that provides required concurrency with minimum number of memories. In order to find out the best memory allocation given a memory configuration, a graph  $G = (V, E)$  is constructed as follows:

$v_{array} \in V$  for each array in the kernel netlists,  $size(v_{array})$  is the size of the array.

$e_{array_i, array_j} \in E$  if  $array_i$  and  $array_j$  need to be accessed concurrently in any kernel

Assuming that  $N$  memories of size  $S_1, S_2, \dots, S_N$  are specified in the memory configuration. Finding an  $N$  way partition for  $V$  with maximum edge cuts gives us the best memory allocation. This NP-Complete problem is solved using simulated annealing. Using this method, the designers can use the number of cuts as guidance to decide on a desirable memory configuration.

### 3.5.2 Reconfigurable Interconnect Architecture Evaluation

The reconfiguration model depicted in Fig. 4 requires the underlying interconnect architecture to support sufficient concurrency within each kernel configuration, while allowing time-sharing of resources across all kernel configurations. An efficient interconnect network hence must have sufficient flexibility to support the required interconnect patterns, while still maintaining good performance and energy efficiency for every configuration.

A survey of interconnect architectures applicable to single-chip VLSI implementation is given in [25]. In our methodology, we evaluate the three most promising interconnect architectures (shown in Fig. 9).

- **Multi-Bus (Fig. 9a):** This architecture provides full connection flexibility and has the advantage of simple implementation, but it suffers from a large area overhead (due to the large number of global buses) and a high energy consumption (due to the long global buses and the large number of switches).
- **Irregular Mesh (Fig. 9b):** In order to optimize local connections between neighboring modules, an irregular mesh interconnect structure extended from FPGA is proposed. Given a module placement, wiring channels are created along the sides of each module and switch boxes

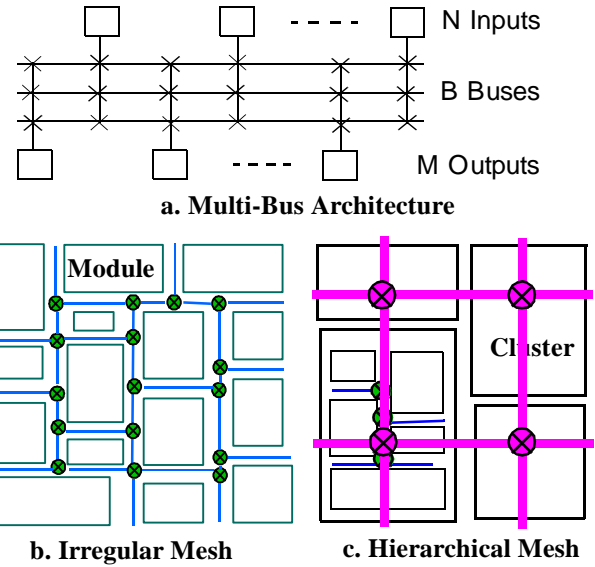


Figure 9. Interconnect Architectures Evaluated in the Maia Design

are placed at channel intersections. The advantage of a mesh structure is that it can provide low-cost connections between local modules. But long connections between distant modules suffer from the large number of switches in the connection, which results in high energy and delay cost.

- **Hierarchical Mesh (Fig. 9c):** While continuing to exploit the locality of interconnections, adding hierarchies to the interconnect network can reduce the number of switches in long connections. Based on the interconnect requirements extracted from applications (stage 3), one can partition the system into many clusters of tightly-connected modules (Fig. 9). Within each cluster, an irregular mesh network provides the intra-cluster connections. A second larger-granularity mesh is superimposed on top of this local network to provide energy-efficient and high-performance inter-cluster connections. Obviously, this architecture can be extended with more levels of hierarchy.

All the implementation parameters of the above three architectures are evaluated and optimized iteratively with the help of a graph-based router which can route each net in a kernel on the underlying network architecture to predict the energy and delay cost. In the router, each interconnect network architecture is described as a graph with the vertices representing the ports and switches and edges presenting feasible connections. Finding an optimal route between two ports is formulated as finding the minimum Steiner tree on the graph and a heuristic algorithm proposed by [31] is employed to solve this NP-complete problem. The more sophisticated interconnect cost model is used to compute the

weights for each connection. The final cost of a net is computed as the total cost of the segments on the route. All cost informations are propagated up and incorporated into the kernel macromodels.

### 3.5.3 Efficient Configuration Code Generation

On the software implementation side, while the program body that does not belong to kernels is compiled down to the processor using standard compilation tools, it is a big challenge to generate compact configuration code for a computational kernel. Depending upon if the configuration code can be generated during compile time or runtime, we define it as static or dynamic, respectively. Obviously, it is more desirable to execute static codes. Therefore, we divide configuration codes into three categories. Each categories has its unique characteristics that help us determine if it favors static or dynamic configuration:

- Interface codes: These codes take care of synchronization between the processor and satellites, including resetting of all satellite processors and network before kernel execution; data transfer between processor and satellites etc. These codes are static (known after memory allocation).
- Configuration code for the satellites: For a given kernel, most of the operations to be performed on satellites are known and therefore are static generated. The only exception is the address generator which can generate data of different length which sometimes can only be determined at runtime.
- Configuration code for the interconnect: It is very time consuming to perform runtime routing. Therefore, the netlist is routed on the underlying architecture using the router described in 3.5.2 at compile time.

Since the intermediate form has all informations on satellite functionality and connectivities, it is equipped with automatic code generation capability.

More optimizations such as caching of configuration codes and partial reconfiguration to improve configuration latency are part of our future research[32].

## 4. CASE STUDY END RESULTS

In the following case study implementations, all processor and satellite modules (embedded ARM8 core, MAC, ALU, memory, address generator, FPGA satellite) are implemented in 0.25 um technology and all energy and performance of these modules are characterized using the methods described in Section 3.3. Powers of each of the case studies are estimated using our methodology also and presented in this section.

## 4.1 Kernel Explorations

We first illustrate high level algorithm and architecture design exploration for computational kernels by mapping a multiuser detection LMS filter to the Pleiades architecture. The specification of the multiuser detection algorithm, documented in [33], provides symbol rate at 1.67 MHz for a spreading factor of 15. The computations required by the filter are mainly dataflow with limited control flows (looping) which is suited for computations on the Pleiades satellites very well.

The C specification of the algorithm is profiled and the multiplication, memory and alu operation counts are 300MOPS, 640MOPS and 320 MOPS respectively. This first order profiling information tells us that existing programmable processor probably can not meet the required computational power. Since the profiling information also indicated that high memory operations are needed, we run the algorithm through copy propagation optimization and got a refined algorithm with 37.5% less of the original memory operations.

We perform a direct-mapping of the refined LMS algorithm. The mapping is simulated in the intermediate form and timing and energy data is gathered. The estimated power and area of the Pleiades implementation are shown in the second row of Table 1. For the other two implementations of this algorithm (ASIC and TMS320C54x), the method to obtain the power and area information is described in [33]. This estimation allows the designers to get fast feedback on different architectural choices (Table 1) at the early stage of the design cycle.

Architecture	Power (mW)	Area (mm <sup>2</sup> )
TMS320C54x	460 * [33][19]	1089
Pleiades	<b>18.04</b>	<b>5.07</b>
ASIC	3 [33]	1.5

Note\*: TMS320C54x requires 36 DSPs in parallel which is unrealistic

Table 1: Architecture Comparison for LMS filter

## 4.2 Maia Design and Results

Many complicated DSP and communication algorithms (e.g. voice and video CODECs) contain not only dataflow kernels, but also higher level controls. We demonstrate snapshots of the exploration methodology in dealing with both control and dataflow computations, as used in designing a low-power baseband speech processing chip (Maia). The mapping of the 16-bit fixed point VSELP encoder is used as the demonstration vehicle. The specification

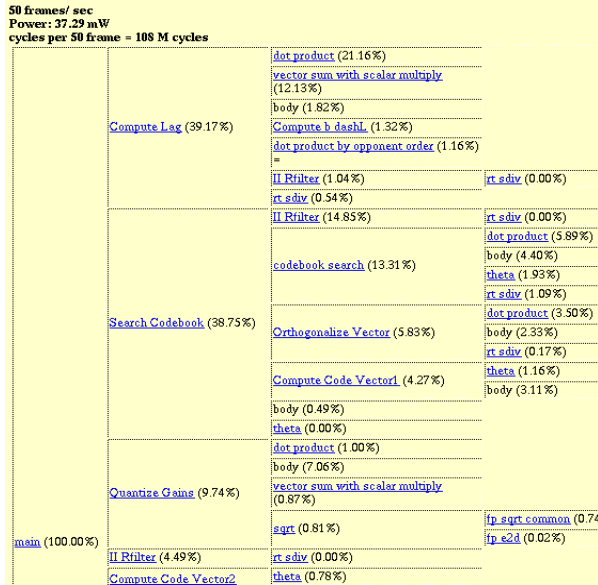


Figure 10. ARM8 Energy Percentage Breakdown for

requires the VSELP encoder to process 50 speech frames per second.

The algorithm is specified in C. All dataflow kernels have already been extracted into procedure calls in this example.

#### 4.2.1 Software Based Approach

The algorithm is compiled to the specified microprocessor and simulated using the ARM instruction-level energy profiler. For the baseline cost, the core is set to run at 2.5V, 120 MHz to meet the timing constraint. The total energy, timing, and the energy percentage breakdown of each function is listed in Fig. 10. The hierarchical call graph is constructed as a hierarchy of macromodels (Fig. 6). At this stage, each node is given an initial cost based on software implementations, which is a function of the parameter VOLTAGE since the energy of the software is proportional to the core speed.

Based on the energy percentage breakdown in our example, the top 3 energy consuming kernels are:

- dot\_product in ComputeLag: 21.16%
- IIRfilter in SearchCodebook: 14.85%
- vector\_sum\_with\_scalarmul in ComputeLag: 12.13%

The next step is to start moving kernels of high rankings to Pleiades satellites.

#### 4.2.2 Kernel Mapping and Partitioning

The mapping and the macromodel of the dot\_product kernel discovered at the kernel ranking step is shown in Fig. 11. The parameter Count is the number of times the

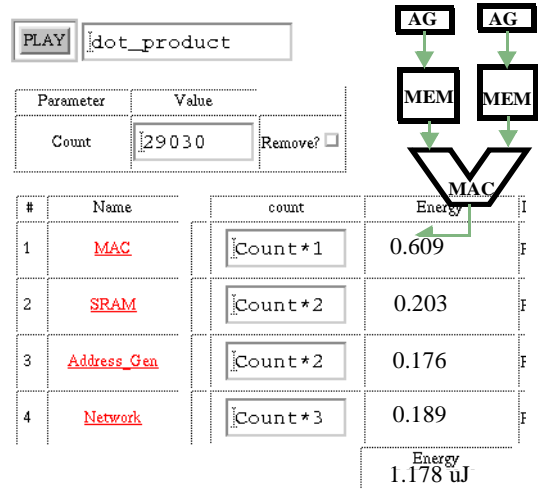


Figure 11. A kernel macromodels viewed in PowerPlay

kernel is invoked in the algorithm (determined by the profiler in stage 1). This kernel macromodel is used to replace the node that corresponds to the dot\_product function in ComputeLag so the total energy cost can be re-evaluated.

We then iterate the mapping and re-evaluation process, going down the kernel ranking list. For the VSELP algorithm, the final partitioning has 79.35% (in terms of energy) of the software-only computation implemented as kernels on the Pleiades satellites and the rest on microprocessor. In terms of functionality, the kernels are dataflow loops such as: dot\_product, FIR, IIR, vector\_sum with scalar multiply, ComputeCode, etc.

One of the kernels (ComputeCode) requires computations (theta function) that can not be implemented with ASIC satellites and this computation is mapped to FPGA satellite.

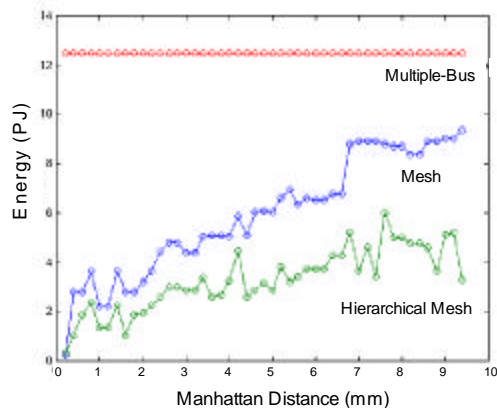
#### 4.2.3 Implementation Optimization

##### Architecture Instantiation

For most of the computation satellites, the IIR kernel sets the upper bound of number of modules needed in the voice coding architecture (2 MACs, 2 ALUs). In addition, a 4x8 FPGA satellite module is also needed to implement the theta function. Two memory configurations were under consideration: four 2K word memories or four 1K and four 512 memories. After allocating arrays (from all VSELP kernels) onto these two memory configurations, it is determined that the second configuration offers the most concurrency and good utilization of final chip area. A floorplan of the architecture instance is shown in Fig. 12.

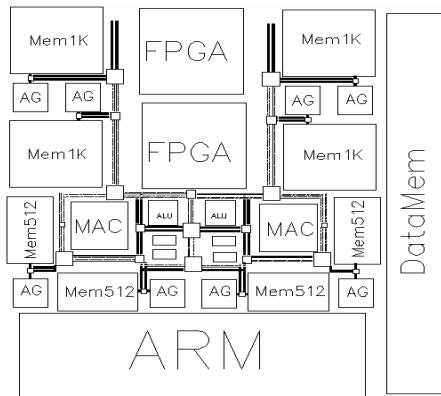
##### Reconfigurable Interconnect Architecture Evaluation

Based on the architecture instance and all the kernels needed to be implemented on the architecture, we explore the effect of interconnect architecture on the overall application performance to determine an optimal reconfigurable interconnect structure. The router in section 3.5.2 is used to route all kernels onto all three interconnect architectures - multibus, mesh and hierarchical mesh. Fig. 13 shows the energy statistics of all module to module connections of the three interconnect architectures. As predicted, the hierarchical mesh is the only architecture that can optimize both local and global connections. We then use the module to module information as the interconnect cost in all kernel macromodels and obtained results in Fig. 14.

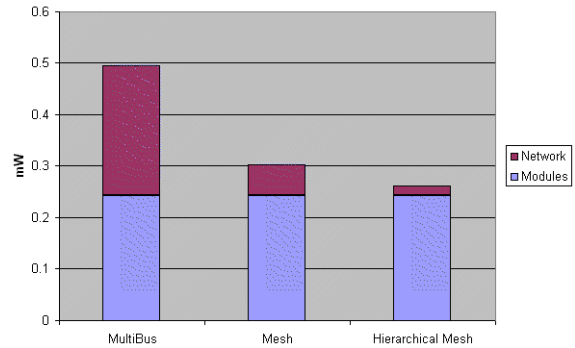


**Figure 13. Module-to-module connection energy of different architectures against manhattan distance**

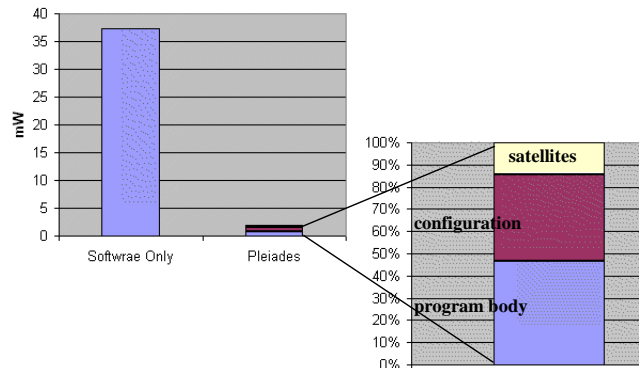
The results show that the hierarchical mesh structure has the most energy efficiency (an order of magnitude of improvement over simple multi-bus implementation) and is therefore chosen as our implementation of choice.



**Figure 12. The Maia Architecture**



**Figure 14. Impact on the Overall Application Energy**



**Figure 15. Final Energy Breakdown for the VSELP algorithm**

#### 4.2.4 Final Results

Using the intermediate form specification of the kernels, configuration codes are generated. The total number of instruction cycles (program body and configuration codes) needed to run on ARM only requires the processor to run at 1V, therefore the parameter VOLTAGE is changed accordingly. The final energy of the VSELP algorithm implemented on Pleiades (including the ARM8 power) is shown in Fig. 15. By partitioning the algorithm across processor and Pleiades satellites, more than an order of magnitude of improvement (~20 times) in energy efficiency is achieved.

### 5. CONCLUSION AND FUTURE WORK

A methodology for heterogeneous reconfigurable DSP architectures is presented in this paper. The design environment allows design explorations at different abstraction levels (algorithm, hardware and software architecture) and provides a bridge from the algorithm to an energy efficient implementation on a reconfigurable platform.

Automatic mapping and more configuration code optimizations (configuration code caching, partial configuration

etc.) are in progress. Our current architecture research also focuses on more advanced reconfigurable interconnection networks and the design environment will be extended to support them.

## 6. ACKNOWLEDGEMENTS

We would like to acknowledge DARPA's support for the Pleiades project (DABT-63-96-C-0026). The authors would like to thank Seno Katsunori and Yuji Ichikawa for their early work on the Pleiades prototype and evaluation. We would like to acknowledge other members on the Maia design team. We would like to thank Ning Zhang for her help on the analysis of the multiuser detection algorithm.

## 7. REFERENCES

- [1] G. R. Goslin, "A Guide to Using Field Programmable Gate Arrays for Application-Specific Digital Signal Processing Performance", *Proceedings of SPIE*, vol. 2914, p321-331.
- [2] M. Goel and N. R. Shanbhag, "Low-power equalizers for 51.84 Mb/s very high-speed digital subscriber loop [VDSL] modems", *Proceedings of IEEE Workshop on Signal Processing Systems*, Oct. 1998, Boston.
- [3] The Pleiades project homepage (<http://info-pad.EECS.Berkeley.EDU/research/reconfigurable/>)
- [4] A. Abnous and J. Rabaey, "Ultra-Low-Power Domain-Specific Multimedia Processors", *Proceedings of the IEEE VLSI Signal Processing Workshop*, San Francisco, California, USA, October 1996.
- [5] A. Abnous et al., "Evaluation of a Low-Power Reconfigurable DSP Architecture", *Proceedings of the Reconfigurable Architectures Workshop*, Orlando, Florida, USA, March 1998.
- [6] J. Borel, "Technologies for multimedia systems on a chip", *1997 IEEE International Solid-State Circuits Conference*, pages. 18-21.
- [7] J. M. Rabaey, "Reconfigurable Computing: the Solution to Low Power Programmable DSP", *Proc. to 1997 ICASSP Conference*, Munich, April 1997.
- [8] B.P. Dave, G. Lakshminarayana and N. K. Jha, "COSYN: Hardware-Software Consynthesis of Embedded Systems", *Design Automation Conference*, pp.703-708, 1997.
- [9] G.De Micheli and R. K.Gupta. "Hardware/Software Co-Design", pages 349-365, *Proc. of the IEEE*, Vol 85, No.3, March 1997.
- [10] J. Hauser and J. Wawrzynek. GARP: A MIPS processor with a reconfigurable coprocessor. In J. Arnold and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGA for Custom Computing Machines*, Napa, CA, April 1997.
- [11] T. Garverick et al, NAPA1000, <http://www.national.com/appinfo/milaero/napa1000>
- [12] R. Razdan, K. Brace, M. D Smith, "PRISC software acceleration techniques", *Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Cambridge, MA, USA, Oct. 1994
- [13] A. Kalavade and P. A. Subrahmayan, "Hardware/Software Partitioning of Multi-function Systems", *Proceedings of IEEE International Conference on Computer Aided Design*, San Jose, CA, USA, 9-13 Nov. 1997.
- [14] M. Wan, D. Lidsky, Y. Ichikawa, J. Rabaey, "An Energy Conscious Methodology for Early Design Exploration of Heterogeneous DSPS", *Proceedings of the Custom Integrated Circuit Conference*, Santa Clara, CA, USA, May 1998.
- [15] D. Lidsky and J. Rabaey, "Early Power Exploration -- a World Wide Web Application", *Proc. Design Automation Conference*, Las Vegas, NV, June 1996.
- [16] R. Mehra, et al. "Algorithm and Architectural Level Methodologies for Low Power", pages 335-362, in Rabaey (14).
- [17] Stanford Unified Intermediate Form, <http://www-suif.stanford.edu/suif/>
- [18] V. Tiwari, S. Malik, A. Wolfe, and T.C. Lee, "Instruction Level Power Analysis and Optimization of Software", *Journal of VLSI Signal Processing*, 1996.
- [19] Texas Instruments Application Reports, <http://www.ti.com/sc/docs/psheets/appnote.htm>.
- [20] Advanced RISC Machines, ARMulator version 2.10
- [21] P. Landman and J. Rabaey, "Black Box Capacitance Models for Architectural Power Analysis", *Proc. of the International Workshop on Low Power Design*, pp. 165-170, April 1994.
- [22] V. George, H. Zhang, J. Rabaey, "Low Energy FPGA Design", submitted to ISLPED 1999.
- [23] T. Callahan P. Chong, A. DeHon and J. Wawrzynek, "Fast Module Mapping and Placement for Datapaths in FPGAs", *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays (FPGA '98, February 22-24, 1998)*.
- [24] R. Mehra, "High-Level Estimation and Synthesis Techniques for Low-Power Design", *Doctoral Thesis*, May, 1997.
- [25] H. Zhang, M. Wan, V. George, J. Rabaey, "Interconnect Architecture Exploration for Low Energy Reconfigurable Single-Chip DSPs", *Proceedings of the WVLSI*, Orlando, FL, USA, April 1999
- [26] M. Potkonjak and J. Rabaey, "Exploring the Algorithmic Design Space using High Level Synthesis", *Proc. of IEEE Workshop on VLSI Signal Processing VI*, pp.123-131, 1993.
- [27] SunSoft Press, "Solaris Multithreaded Programming Guide".

- [28] A. Kalavade, "System Level Codesign of Mixed Hardware-Software System", *Tech Report UCB/ERL 95/88*, Ph.D. Dissertation, Department of EECS, University of California, Berkeley, CA 94720, September 1995.
- [29] E. Kusse, "Analysis and Circuit Design for Low Power Programmable Logic Modules", *Master's Thesis*, UC Berkeley, 1997.
- [30] S. Hauck, "Configuration Prefetch for single context reconfigurable coprocessors", *Proceedings of 1998 International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, 22-25 Feb. 1998
- [31] M. J. Alexander, et al. "Performance-Oriented Placement and Routing for Field Programmable Gate Arrays", *Proceedings of EURO-DAC*, European Design Automation Conference, Brighton, UK, 18-22 Sept. 1995.
- [32] S. Li, M. Wan and J. Rabaey, "Configuration Code Generation and Optimizations for Heterogeneous Reconfigurable DSPs", *Proceedings of SiPS*, 1999.
- [33] N. Zhang, "Implementation Issues in a Wideband Receiver Using Multiuser Detection", *Master's Thesis*, University of California at Berkeley, 1998