An Architecture Exploration Framework for the Implementation of Embedded DSP Applications

by

Ahmed Elhossini

A PhD proposal

presented to the University of Guelph

for the degree of

Doctor of Philosophy

in

Systems & Computer Engineering

Guelph, Ontario, Canada, 2007

Abstract

Advances in chip technology have enabled integrating many functional units on a single chip. This led to the emergence of the concept of System-on-Chip (SoC). SoC is the foundation for the development of advanced embedded systems. Embedded systems are widely used today in different Digital Signal Processing (DSP) applications that usually require high computation power and tight constraints. Using SoC technology increases the challenges facing the designer to choose the optimal design. A tool that helps explore different architectures is required to design an efficient system. The tool should be able to explore different architectures and evaluate them according to the given constraints. The design space to be explored depends on the application domain, and the target platform. Reconfigurable devices, such as Field Programmable Gate Arrays (FPGA), have evolved to the extent that a complete DSP application can be implemented on a single device. Due to the variety of architectures and different objectives that constrains the design of SoC embedded systems, Architecture Exploration (AE) could be viewed as a multi-objective optimization problem.

In this proposal different approaches for architecture exploration are reviewed. Multi-Objective Evolutionary Algorithms (MOEA) have proven to be an efficient technique in exploring the design space of DSP embedded systems. An implementation of an architecture exploration framework based on MOEA and analytical evaluation of embedded system is proposed. The design space is defined by using an experimental core library. Each item in the library is represented as a set of attributes used for system evaluation. Preliminary results indicate that the proposed approach is valid for solving the architecture exploration problem.

Based on the preliminary results and the literature review, the following are the recommendation for future work: i) Enhance the evaluation model; ii) Incorporate run-time reconfiguration to enable implementing large applications on a single reconfigurable design; iii) The design of DSP oriented reconfigurable device.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past few decades the demand for embedded Digital Signal Processing (DSP) systems has been increasing constantly. These systems are used in several applications such as MP3 players, wireless communication sets and intelligent hearing-aid devices. Due to the nature of these devices they are usually implemented using System on Chip (SoC) technology. DSP applications are complex, parallel in nature, and time consuming. The designers are usually faced with different conflicting design objectives such as low power, low cost, high flexibility and high performance. For this reason SoC embedded systems have a heterogenous multi-processor architecture in which different components are integrated on a single chip. These components range from fully programmable processors to dedicated hardware blocks. The designer has to select the proper components to optimize the different design objectives. Fully programmable processors could be selected for flexibility, by supporting multiple applications and system extension while dedicated hardware accelerators are used to optimize hard constraints such as time and power dissipation.

Reconfigurable devices, in the form of Field Programmable Gate Arrays (FPGA), are becoming more and more attractive in implementing digital systems. Modern FPGAs have high logic density and are equipped with advanced digital blocks in the form of embedded multipliers, DSP blocks and embedded processors. This enables FPGAs to be a suitable implementation platform for SoC based embedded systems. Coarse-Grained Reconfigurable Arrays (CGRA) are also another form of reconfigurable devices. Their architecture is a compromise between FPGAs, that have high level of reconfigurability with the cost of more chip area, power consumption and speed, and ASIC implementations, that lacks flexibility.

Several architectures are available to implement a given DSP application using reconfigurable devices. Selecting a suitable sub-optimal architecture for the given application is a very challenging problem. A tool that helps the designer to select the optimal architecture is of great interest to reduce development time.

Several studies have investigated the architecture exploration problem and introduced different frameworks for implementing embedded systems [Bech03, Pale04, Khar01, Asci05a, Kim06]. Most of these studies dealt with the exploration of a parameterized platform that can be configured and adopted to a specific problem. However none of these studies dealt with the design of a general architecture with no initial structure. Different optimization and evaluation techniques are investigated in the literature. Selecting an effective optimization technique for architecture exploration is one of the main goals of this research.

1.1 Research Motivations

The main motivations behind this research proposal can be summarized as follows:

- The complexity of the DSP applications: Implementing DSP applications using SoC embedded systems increases the complexity and constraints for the designer.
- The design space is very large due to the availability of several Intellectual Property (IP) cores and embedded processors. Selecting the proper architecture requires experience and knowledge about the design of digital embedded systems. This makes it hard for the designer to select an optimal architecture for the given application.
- Searching the design space for optimal configurations can be formulated as multi-objective optimization problem with conflicting objectives. Solutions obtained offer the SoC designer with a set of sub-optimal configurations (Pareto-optimal set) which he/she can choose from.
- Most DSP applications are usually developed by software engineers who lack hardware design knowledge and experience required for the design of these systems. Software engineers need a tool that narrows the gap between the pure software implementation and SoC implementation.

1.2 A Methodology For Design Exploration

Architecture exploration tools tend to explore the design space to find an optimal or near optimal solution for a given application. During the exploration phase several architectures are generated and evaluated to determine their optimality. Optimality is measured by satisfying the normally conflicting constraints and objectives such as area, performance, power consumption and flexibility. Multi-objective optimization (MOO) techniques can be used for such effective exploration. The literature shows that Multi-Objective Evolutionary Algorithms (MOEA) are efficient and robust to explore the complex design space of heterogenous embedded systems [Erba06]. The resulting architecture includes several components from a core library and the mapping of the application on to the resulting architecture.

The optimality of the generated architectures is then evaluated. Because several architectures need to be evaluated, the evaluation technique used should give quick results with good level of accuracy. An accurate evaluation technique can be performed at a lower level of abstraction for the selected candidates. Analytical evaluation is usually used to give quick results. Analytical models for each component in the architecture is used to estimate the overall performance of the architecture [Niar06]. The evaluation results are fed back to the optimizer to accept or reject the generated architecture as illustrated in Figure 1.1.

The research in this proposal is directed towards the design of an architecture exploration framework for implementing DSP embedded systems on FPGAs and CGRAs. The research in this proposal can be summarized as shown in Figure 1.1:

1. Different meta-heuristic methods will be investigated to select the proper technique to search the design space for the given problem. The list includes evolutionary techniques, simulated annealing, reactive tabu search and random search. This phase will eventually produce an efficient heuristic method to effectively search the design space of the given problem.

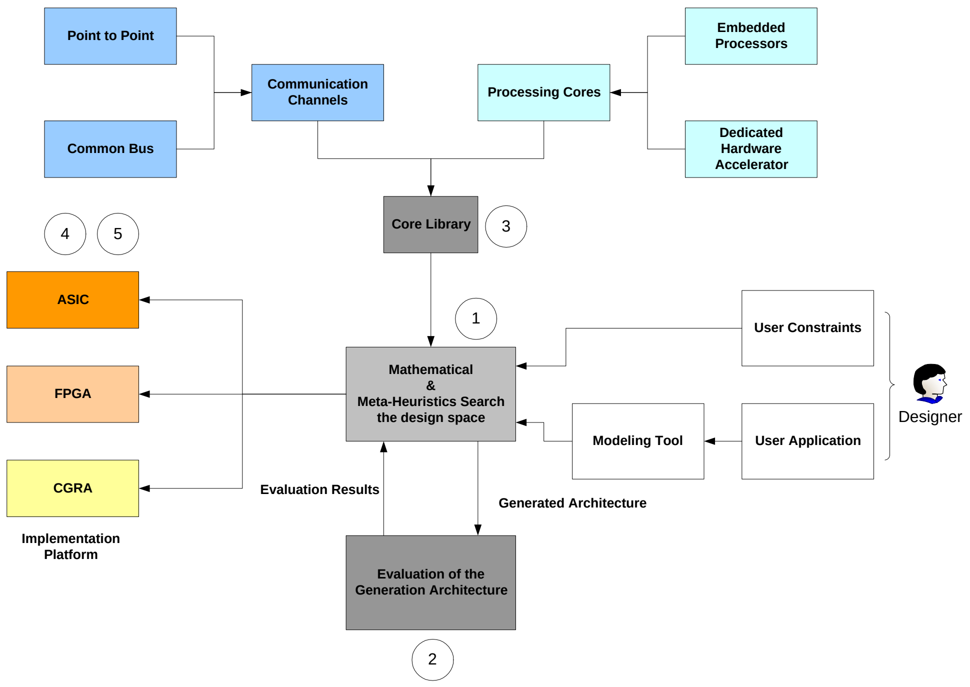


Figure 1.1: AE framework

2. Different evaluation techniques for embedded systems will be examined to develop a suitable evaluation scheme for the framework. The resulting evaluation technique will be combined with the search module to build an effective architecture for the target application.

3. A core library of different architectures will also be developed. It will include specifications of the components that will be used by the framework to create the optimal design. The library should contain basic components ranging from processing cores to communication channels to build a simple DSP application.

4. Investigate the implementation of the resulting architecture on different platforms such as ASIC, FPGA, and CGRA. The goal of this phase is to integrate the proposed framework with the appropriate platform using physical design implemen-

tation tools.

5. In the case of large applications that do not fit in a single FPGA or CGRA, runtime reconfigurability will be an alternative solution to swap unused portions of the design with the required one. The use of runtime reconfiguration will be examined. This requires investigating the runtime switching and scheduling.

1.3 Proposal Organization

The remainder of the proposal is organized as follows: Chapter 2 provides essential background on reconfigurable devices, multi-objective optimization techniques and the architecture exploration process. In chapter 3 a literature review on searching the design space for architecture exploration, embedded systems evaluation techniques, architecture exploration frameworks and CGRA architectures is presented. Chapter 4 proposes an architecture exploration framework based on a modified Multi-Objective evolutionary algorithm along with preliminary results is given. Finally, chapter 5 gives the proposed future work and directions.

Chapter 2

Background

Reconfigurable logic devices are commonly used today as the main processing element of embedded systems. Modern FPGAs contain many resources and embedded blocks that enable implementing a complete system using a single chip (SoC). The design of these systems is a complex task due to the availability of different Intellectual Property (IP) architecture and also the lack of knowledge in the design of these sophisticated digital embedded systems. Architecture exploration tools are required in the early design phases to search the design space for an optimal solution. The architecture exploration problem can be viewed as a multi-objective optimization problem. Several objectives are optimized with given constraints. Many architectures are generated and evaluated during the search process. The evaluation phase is a crucial task in the exploration process.

This chapter gives some background on reconfigurable logic devices, the architecture exploration process, multi-objective optimization, and the evaluation process of embedded systems.

2.1 Implementation Approaches for Digital Systems

The rapid development and advancement in fabricating integrated circuits introduced different approaches to implement Application Specific ICs (ASIC). These approaches are classified into two main categories: full custom ICs, and semi-custom ICs as illustrated by Figure 2.1. In the first category all logic cells and routing circuitry are customized. Designers spend many hours to handcraft and optimize each transistor to implement the chip. This allows the designer to include analog circuits, optimized memory cells, or micro-electro-mechanical systems on an IC. Full-custom ICs are the most expensive to manufacture and design. This approach has a long time-to-market and is therefore intended for specific applications that require a high level of optimization.

On the other hand, in semi-custom ASIC design, all the logic cells are predesigned and some (or all) of the routing circuitry is customized. Predesigned cells from a cell library decreases the design challenges faced in full custom design. Semi-custom ASICs can be further classified to standard-cell base, and gate-array based.

In the standard-cell based ASIC, a library is provided by the vendor. Each cell in the library has different versions optimized for several design objectives. Full custom methods are used to optimize each cell during the design of the cell library. Standard cells are placed in the chip area and the wiring masks are customized by the designer.

Gate arrays are yet another class of semi-custom ASICs. In this design style pre-optimized gates are placed in the chip area. The designer task is to select the gates required and customize the routing for a given application. The class of gate arrays in which the final fabrication is performed by the chip vendor is called masked gate arrays. Programmable gate arrays (also called reconfigurable devices) is another design style

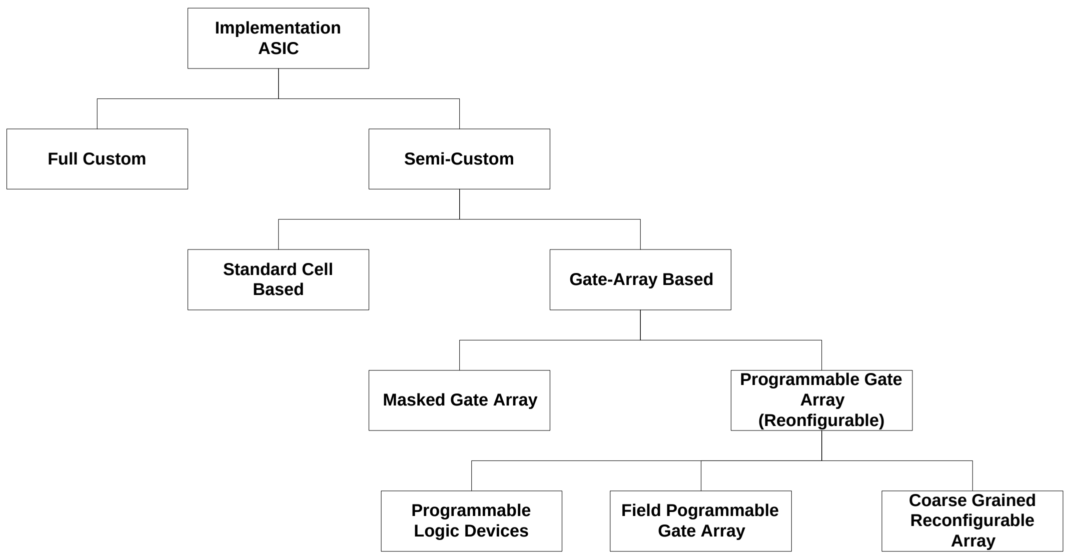

Implementation ASIC
Full Custom
Semi-Custom
Standard Cell Based
Gate-Array Based
Masked Gate Array
Programmable Gate Array (Reonfigurable)
Programmable Logic Devices
Field Pogrammable Gate Array
Coarse Grained Reconfigurable Array

Figure 2.1: Implementation of Digital System using ASIC

where users are able to program the routing and internal configuration of the chip. This class is widely used today for the implementation of embedded systems.

Figure 2.2 shows a simplified flow for the implementation of digital systems. The flow starts by a physical synthesis of the Register Transfer Level (RTL) description of the application combined with the timing constraints. The output of physical synthesis is a netlist with place-and-route information. The netlist combined with the fixed netlist of the predefined components (component library) are used by the place and route back-end tool phase to perform the physical design implementation phase.

2.2 Reconfigurable Logic Devices

Recent development in reconfigurable devices has been possible because of the availability of logic devices that can be rapidly programmed and reprogrammed for different

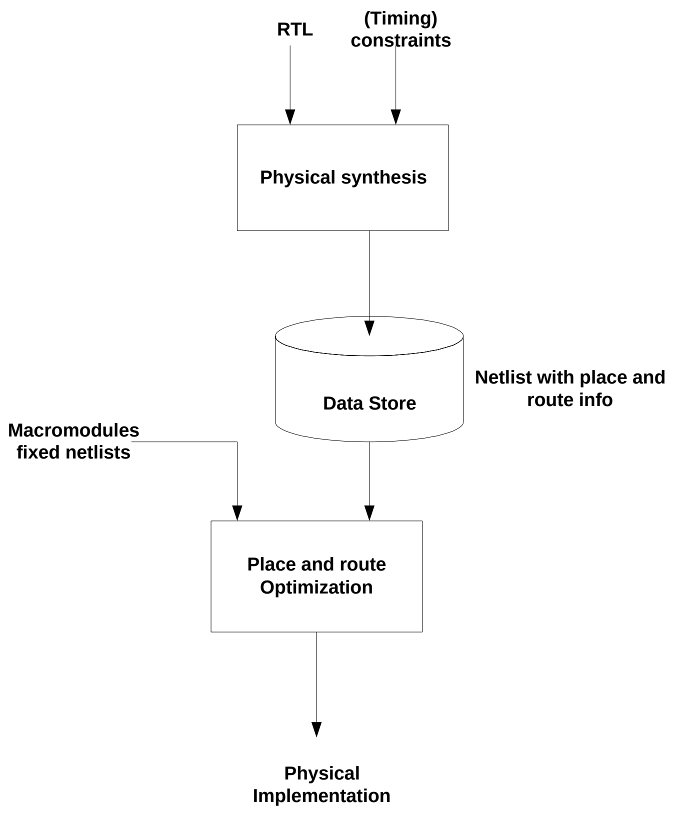

RTL
(Timing) constraints
Physical synthesis
Data Store
Netlist with place and route info
Macromodules fixed netlists
Place and route Optimization
Physical Implementation

Figure 2.2: Implementation Flow of Digital System

applications. The first device introduced from this category of logic devices with reasonable capacity and possibly efficient computation the Field-Programmable Gate Arrays (FPGAs).

FPGAs provide the designer with an array of basic configurable logic blocks usually in the form of Look Up Tables (LUT) and flip flops connected with a programmable interconnection that enables building different functions and memories for a wide range of digital systems. Early generations of FPGAs introduced from different vendors like Xilinx, Altera and others offered relatively fewer logic blocks compared to current generations that provide more resources and tools to build a complete system on a single chip (SoC). Recent FPGA generations also provide more coarse blocks for the designers in the form of embedded multipliers, DSP blocks, multi-gigabit serial I/O and embedded microprocessors that increase the computation power for DSP applications.

2.2.1 Field Programmable Gate Arrays

An FPGA provides the benefits of custom CMOS VLSI, while avoiding the initial development cost, time delay, and inherent risk of a conventional masked gate array. FPGAs are customized by loading configuration data into their internal memory cells. FPGAs can either actively read its configuration data from external serial or byte-parallel PROM (master mode), or the configuration data can be written to the FPGA (slave and peripheral mode). FPGAs can be programmed an unlimited number of times and support system clock rates of up to 500 MHz [Xili06].

An FPGA has three major configurable elements as shown in Figure 2.3:

- Configurable logic blocks (CLB).
- Input/output blocks (IOB).
- Configurable interconnects networks.

The CLB provides the functional elements for constructing user's logic. The IOB provides the interface between the package pins and internal signal lines. The programmable interconnect resources provide routing paths to connect the inputs and outputs of the CLB and IOB onto the appropriate networks. Customized configuration is established by programming internal static memory cells that determine the logic functions and internal connections implemented in the FPGA.

Figure 2.3 depicts an FPGA with a two-dimensional array of logic blocks that can be interconnected by different types of wires. All internal connections are composed of metal segments with programmable switching points to implement the desired routing. An abundance of different routing resources is provided to achieve efficient automated routing. There are four main types of interconnects, three are distinguished by the relative length of their segments: single-length lines, double-length lines and long lines. In addition, buffers drive fast, low-skew nets are most often used for clocks or global control signals.

The principle elements of the CLB are shown in Figure 2.4. Each CLB contains one or more flip-flops and one or more independent n-input function generators in the form of a look-up table. These function generators are very flexible. The CLB may also include a carry logic for fast implementation of arithmetic operations. A CLB implements most

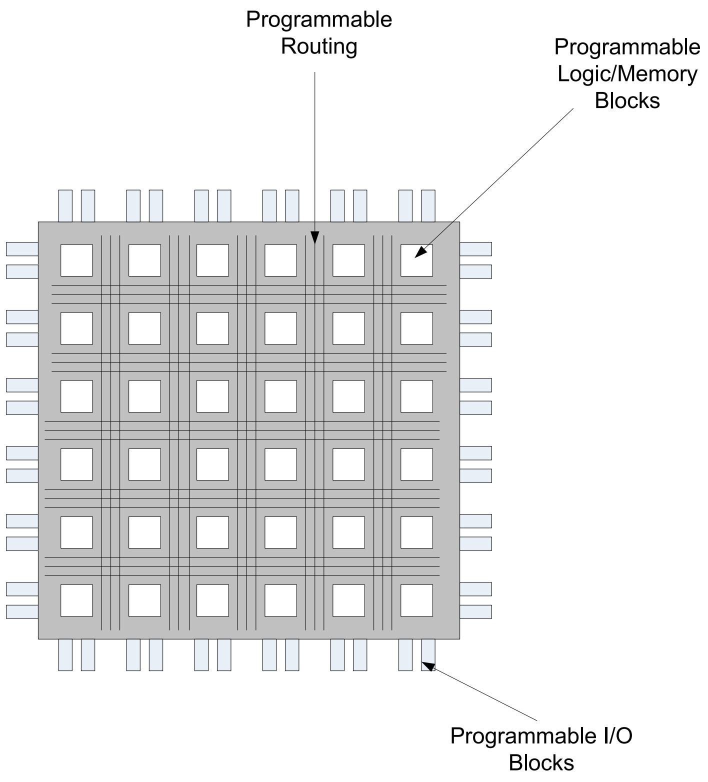


Figure 2.3: FPGA Structure

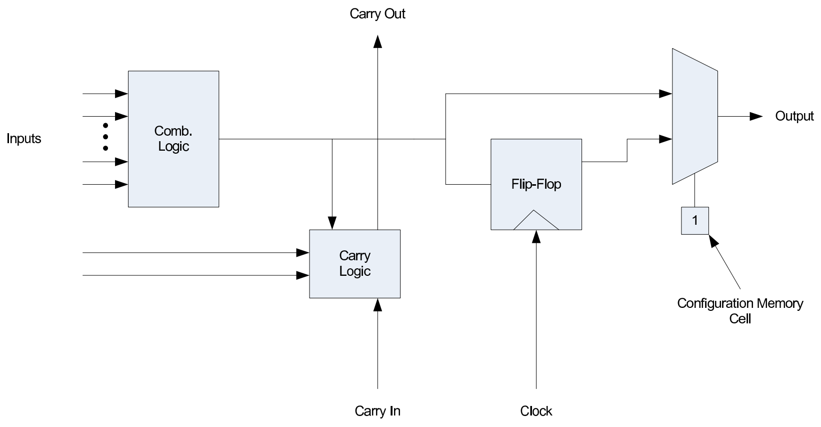


Figure 2.4: General Programmable Logic Block

of the logic in an FPGA. The flexibility and symmetry of the CLB architecture facilitates the placement and routing of a given application [Gokh05].

2.2.2 Runtime Reconfiguration

Runtime reconfiguration in FPGAs enables reconfiguring some portions of the FPGA while the remaining portion operates normally. Modern FPGAs support run-time reconfiguration in several forms [Dont03].

2.2.2.1 Xilinx Virtex Devices

A Virtex FPGA device supports two types of configurations. The entire device can be fully configured or partially configured. Partial configuration enables changing some portions of the device while the remaining portions are operating. The smallest unit that can be loaded into a Virtex device for configuration is called a bitstream “frame”. Each frame is responsible for the configuration of a portion of the device that spans the entire device height and its width normally four reconfigurable blocks (CLBs), which is called a “tile”. There are two styles of partial configuration in Virtex devices; module-based partial reconfiguration and small-bit manipulation.

In module-based partial reconfiguration, distinct portions of an FPGA are referred to as reconfigurable modules. The reconfigurable module can span one or more tiles. The number of reconfigurable modules should be minimal (i.e., a single reconfigurable module) to reduce problems in complex designs. This type of partial reconfiguration is used for independent design applications and for modules that communicate with each other using a special bus macro. Bus macros are responsible of establishing fixed connection points in the design between the reconfigurable and fixed modules [Xili04].

In small-bit manipulations, the partial reconfiguration is accomplished by making a small change to the design, and then generating a bitstream based only on the difference between the two designs. So instead of reconfiguring the entire device, only the portions of the device that changed are configured [Xili04].

In summary modern FPGA devices support runtime reconfiguration. Using a specific procedure, FPGAs can be partially configured. Some portions of the device are configured while the remainder of the device operating normally. This allows large applications to fit in small devices.

2.3 Architecture Exploration

Architecture exploration is the problem of searching the design space of a given application to find an optimal hardware implementation. The application is normally described using a software model and the main objectives of the exploration tool is to construct an architecture and map the software model to the proposed hardware.

The problem of the architecture exploration is illustrated in Figure 2.5. The application is modeled using a set of software blocks modeled in a high level modeling tool or language. The high level description of the application is then converted into a flow graph.

Each block S_i in the graph has several attributes A_i. These attributes might be the block size, power dissipation, speed, etc. The target of the architecture exploration is to specify the cores C_j that construct the proposed architecture and to map block S_i to a

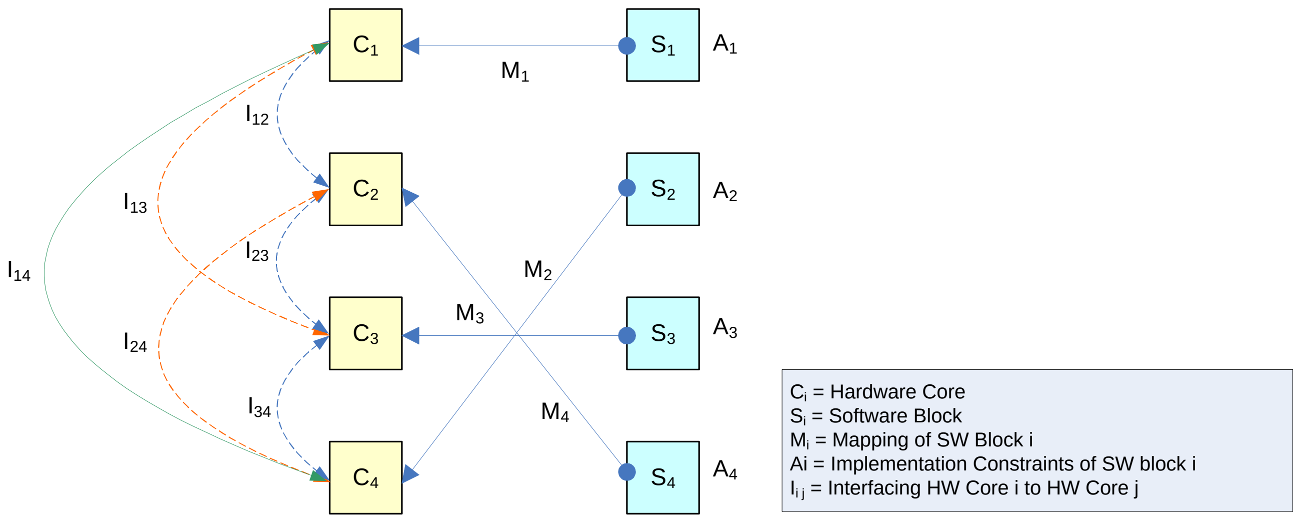

C1
C2
C3
C4
S1
S2
S3
S4
A1
A2
A3
A4
M1
M2
M3
M4
I12
I13
I14
I23
I24
I34
Ci = Hardware Core
Si = Software Block
Mi = Mapping of SW Block i
Ai = Implementation Constraints of SW block i
Ii j = Interfacing HW Core i to HW Core j

Figure 2.5: Problem Definition

specific core C_j. The core C_j can be either a dedicated hardware module, or software code implemented for a specific processor. The core C_j is chosen from a core library. The problem is to find the core C_j, the mapping of S_i into C_j (M_i), and the interfacing between each pair of cores $I_{x,y}$. Each combination of $M_i, C_j, I_{x,y}$ will result in a different architecture. The resulting architectures are then evaluated against the given application constraints. The most appropriate architecture should meet the overall application constraints. An architecture exploration tool is required to explore and evaluate as many designs as possible. The efficiency of the tool is measured by the total numbers of architectures identified, how close the resulting architecture is to the optimal architecture, and the speed of the search process.

The evaluation of the generated architectures is the most important phase of the exploration process. Accurate evaluation will efficiently guide the exploration towards the optimal solution. However, this might be a time consuming process. Evaluation can be performed at different levels of abstraction. Each level provides a different accuracy measure ranging from the transistor level (more accurate, and more complex), to system level (less accurate, but more simple) [Giva02b]. At each level of abstraction, a model should be provided for each core. This model contains information about the power consumption, performance, and area. These models form the core library that is used during exploration.

Architecture exploration is used to perform system-level design for certain application as shown in Figure 2.6. Architecture exploration is added to the digital implementation flow shown previously in Figure 2.2 to reduce the complexity facing the designer to build complex DSP systems. The system level design phase takes a system model from the designer and generates the RTL model and the required timing constrains.

2.3.1 The Y Chart

The Y-chart shown in Figure 2.7 is a general scheme for the design of programmable architectures. This scheme can be used as the foundation of the architecture exploration problem [Liev01].

The main advantage of this scheme is that it separates the architecture from the application. The application is profiled and processed into software blocks. Mapping of the application blocks into the architecture is then performed. The result is evaluated and the architecture, the mapping, and the application are modified if required. This scheme was the foundation of the concept of platform-based design [Keut00].

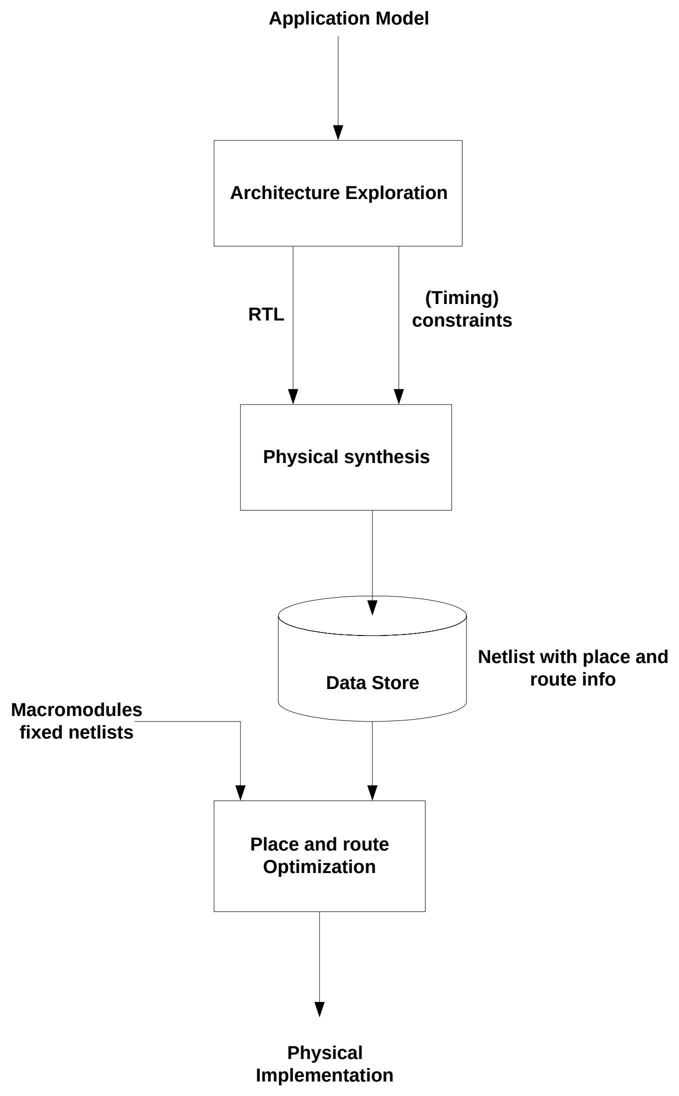


Figure 2.6: Implementation Flow of Digital System With Architecture Exploration

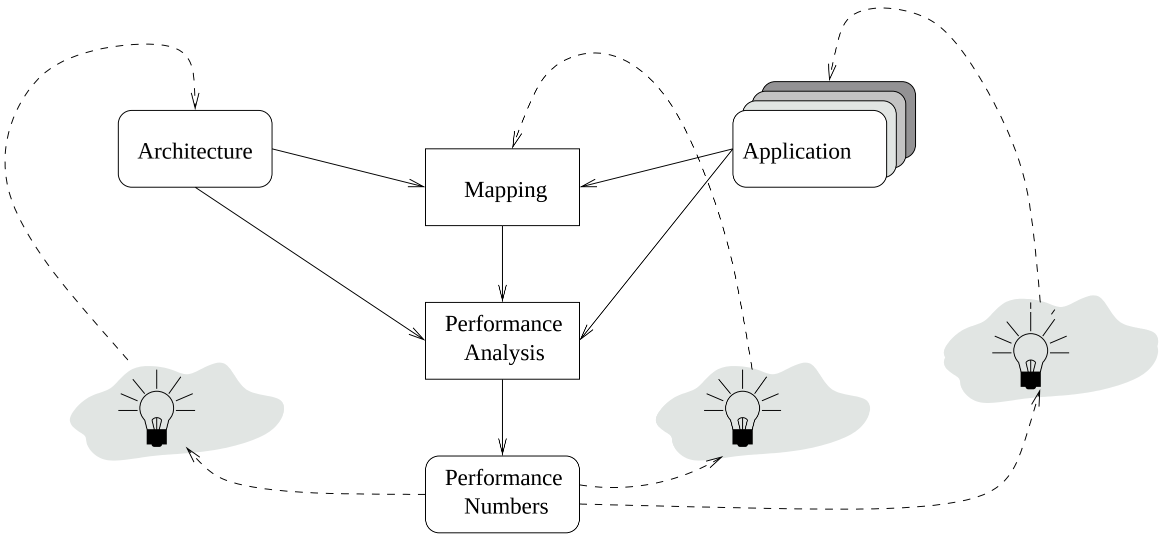


Figure 2.7: The Y-chart: a general scheme for the design of programmable architecture

2.3.2 Architecture Exploration and FPGA

FPGAs include several components that enable building complex systems. Embedded processors, multipliers, memory, and fast I/Os enable implementing multiprocessor systems on a single FPGA. Therefore, FPGAs are considered to be an appropriate platform for architecture exploration. Many cores are available for FPGAs that cover almost every requirement for any DSP application. These cores range from embedded memory, DSP blocks and hard-core processors, to soft-cores such as, MicroBlaze, LEON2, and OpenRisk 1200 [Matt04]. The availability of such variety of cores and soft-processors increases the number of choices available to the designer. An efficient architecture exploration tool is therefore required to select the most appropriate modules for any given application.

2.3.3 Evaluation of Embedded Systems

Evaluation of embedded systems plays an important role in architecture exploration process. It guides the search towards the optimal design that meets the user constraints. The goal of the evaluation process is to extract performance measures of the evaluated architecture. These measures can include speed, area, and power consumption. Based on these measures an architecture can be accepted or rejected.

In general the evaluation of embedded systems can be classified under three categories, as shown in Figure 2.8. Simulators that perform cycle accurate simulation of the processor and peripherals tend to give an accurate evaluation at the expense of huge CPU time. Statistical simulators on the other hand, use statistical information gathered from the profiled and estimated running time of the application on the given architecture. This type of simulator gives a good evaluation in reasonable time. In the analytical evaluation scheme analytical models exist for each computational unit in the embedded architecture. Analytical evaluation gives fast evaluation of a given architecture, with a low level of accuracy. The accuracy can be enhanced to represent a more realistic environment at the expense of more computation time. This evaluation scheme is more suitable for AE tools as a large number of architectures should be evaluated in a short amount of time. Cycle accurate simulation can be used in later stages when more accuracy is required for fine tuning of the resulting architecture.

2.4 Optimization Algorithms

The goal of optimization algorithms is to find the “optimal” or “near optimal” solution among a finite or infinite number of possible solutions, which is achieved by minimizing

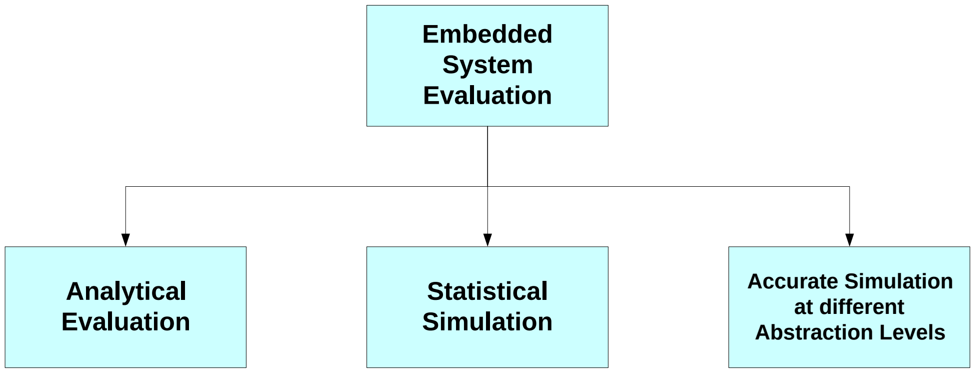

Embedded System Evaluation
Analytical Evaluation
Statistical Simulation
Accurate Simulation at different Abstraction Levels

Figure 2.8: Evaluation of Embedded Systems

or maximizing an objective function.Different techniques can be used to find a solution to the optimization problem. The literature shows that architecture exploration is an NP-complete problem [Asci05a] and therefor, an optimal solution cannot be obtained in polynomial time. The designer can choose to obtain a quick solution at the risk of obtaining sub-optimal solutions. A global optimal solution can be obtained through exhaustive search.

Heuristics methods are used to find a quick sub-optimal solution for the optimization problem. Simulated annealing, tabu search and genetic algorithms are a few examples of such meta heuristics. The goal of these methods is to quickly obtain a near optimal solution by avoiding local minima as shown in Figure 2.9.

2.5 Multi-objective Optimization

Multi-objective optimization is defined as the problem of finding the parameter vector X to optimize a set of objective functions $f_1(X), f_2(X), ..., f_n(X)$. Optimality in multi-objective optimization is to find a solution that gives an acceptable value for the different

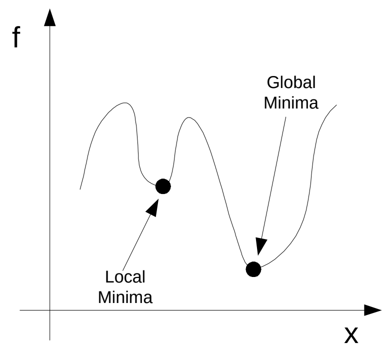


Figure 2.9: Local and Global Minima

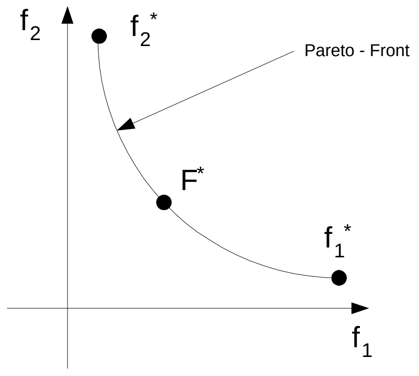


Figure 2.10: Pareto Optimality for Two-Objective Optimization

objective functions compared to the application requirement. Since the objective functions are normally conflicting, several solutions exist for a given problem that meet the given requirement. The boundary of visible solutions in the solution space is called the Pareto-front [Pare96]. The solution is called Pareto Optimal if it falls within the Pareto-Front as shown in Figure 2.10.

The architecture exploration problem can be viewed as a multi-objective optimization problem. The architecture exploration tool should search the design space for the given application to find a Pareto-Optimal architecture. Many heuristic multi-objective

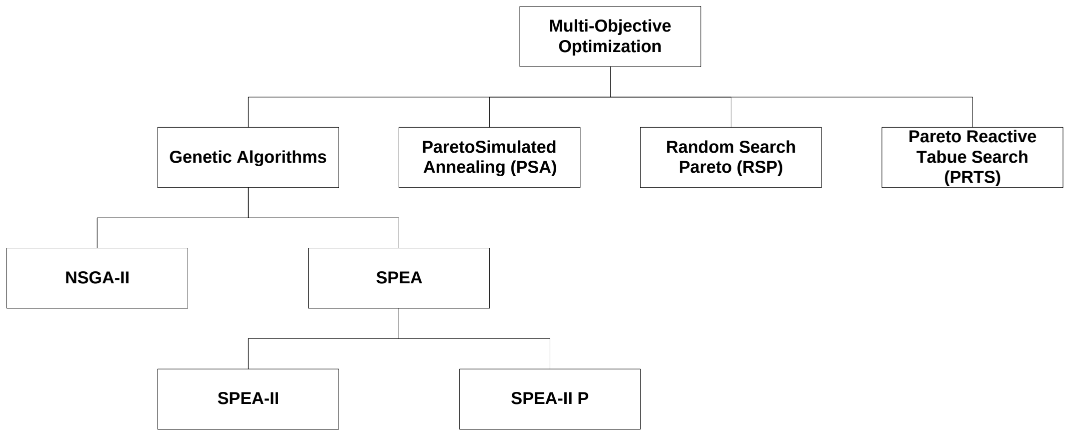

Multi-Objective Optimization
Genetic Algorithms
ParetoSimulated Annealing (PSA)
Random Search Pareto (RSP)
Pareto Reactive Tabue Search (PRTS)
NSGA-II
SPEA
SPEA-II
SPEA-II P

Figure 2.11: Heuristic Multi-objective optimization

optimization approaches are used for efficient architecture exploration as shown Figure 2.11. These approaches will be discussed next.

2.5.1 Pareto Simulated Annealing (PSA)

Simulated Annealing (SA) is a MontCarlo approach for minimizing objective functions [Suma04, Suma02]. In the simulated annealing algorithm a new configuration is constructed by generating a random displacement. If the cost function of this new state is better than the previous one, the change is accepted, but if it is worse, the new configuration is accepted with a certain probability. The Pareto Simulated Annealing (PSA) is the multi-objective version of SA. At each step the starting point is a set of configurations and not a single one.

2.5.2 Pareto Reactive Tabu Search (PRTS)

The Pareto Reactive Tabu Search (PRTS) [Batt94] is the multi-objective version of the Tabu Search (TS) algorithm. The key concept of the algorithm is the tabu list which contains prohibited moves that consists of the most recently visited configurations. The purpose of this list is to avoid falling in a local minima. The PRTS is an evolution of the TS algorithm but it employs an adaptive prohibition period and escape mechanism to support the tuning of a multi-objective problem.

2.5.3 Genetic Algorithms

Genetic Algorithms (GAs) were introduced by John Holland in the 1960s in the University of Michigan in the 1960s and 1970s. Holland's original goal was not to solve a specific problem, but to formulate the adaptation phenomenon as it occurs in nature and to develop algorithms that can be used to import natural adaptation into computer systems [Mitc96]. Genetic algorithms are considered today a class of optimization algorithms. Genetic algorithms are based on moving from one population of "chromosomes" (represents a set of initial solutions) to a new population by using a kind of "natural selection" together with the genetics-inspired operators of crossover, and mutation. The chromosome is represented by a set of "genes" (each gene represents a parameter of the solution). The operators used in genetic algorithms can be summarized as follows:

- The selection operator, chooses individuals (chromosomes) from the population that will be allowed to reproduce. Each individual is evaluated to measure its fitness. The most fit individuals are allowed to produce more offsprings. The selection is made with the hope that the new offsprings will be more fit than their parents.

- The crossover operator, exchanges subparts of two individuals. This operation imitates the biological recombination between two chromosomes.

- The mutation operator, randomly changes the value of some genes in the chromosome. This operation resembles mutation in the gene structure of living organisms caused by the surrounding environment. This allows genetic algorithms to perform random jumps in the search space.

Starting by an initial population (initial solutions), using the selection operator certain individuals are selected to perform crossover and mutation to generate a new population. The new generations are assumed to be closer to the optimal solution than older generations. The process is repeated until the target solution is found or a specific number of generations is reached.

2.5.4 Multi-Objective Evolutionary Algorithms (MOEA)

Multi-objective evolutionary algorithms are the class of genetic algorithms that are used for the optimizations of different conflicting objectives. In multi-objective optimization the fitness of each individual should reflect the effect of each objective, which means it should depend on the different objective functions. Different algorithms exists with different fitness assignment schemes to solve the multi-objective optimization problem. Fitness assignment aims to calculate a single fitness value for an individual from the different conflicting objectives functions.

There are two commonly used multi-objective genetic algorithms found in the literature, SPEA and NSGA-II.

2.5.4.1 Strength Pareto Evolutionary Algorithms (SPEA)

Strength Pareto Evolutionary Algorithms (SPEA) is a multi-objective evolutionary optimization algorithm developed by Zitzler et al [Zitz99]. The algorithm was then modified to SPEA2 in [Zitz01, Zitz02]. Another modification was made to the approach to improve its search capabilities in [Kim04]. The algorithms uses the concept of dominance to assign fitness values to individuals. It does so by taking into account the number of individuals a solution dominates and is dominated by according to Pareto optimally. The algorithm makes use of an external set (archive) that hold the most fit non-dominated solutions across all generations. Distinct fitness assignment schemes are defined for the generation population and the archive to always ensure that better fitness values are assigned to individual in the archive. It uses binary tournament with replacement. Selection is made only from the archive which contain the non-dominated solutions. Solutions generated from SPEA2 may require a repair phase if they are infeasible.

2.5.4.2 Non-Dominated Sorting Genetic Algorithm (NSGA)

The Non-dominated Sorting Genetic Algorithm (NSGA) [Srin94, Deb02] is another multi-objective evolutionary algorithm. Similar to SPEA, it makes use of the concept of dominance to assign fitness values to the solution. It uses a more complex fitness assignment scheme based on sorting the population into groups depending on the dominance of each individual. The algorithm is modified by [Deb02] to NSGA-II for better fitness assignment and faster sorting.

In [Erba06] a comparison is made between SPEA2 and NSGA-II in the field of AE of general embedded systems. The results show that NSGA-II is superior to SPEA2 in most of the test benches used. They gave almost the same results compared to the Pareto optimal solutions, with NSGA-II being two times faster.

2.5.4.3 Other Multi-Objective Evolutionary Algorithms

There are other MOEA found in the literature, such as MOGA [Fons93], NPGA [Horn94], PEAS [Know99], and Rudolph elitist GA [Rudo01]. MOGA, and NPGA share some features. Both assign fitness to individuals based on non-dominated sorting, and both preserve diversity among solutions of the same non-dominate level. SPEA-II and NSGA-II are considered an evolution of these algorithms. PEAS use a single parent- single off-spring EA strategy. A binary string is used to represent individuals. Starting from a single parent, binary mutation is used to generate an off-spring, which is then compared to its parent. If the off-spring dominates the parent, the parent is discarded and the off-spring is used as a parent in the next iteration. If the parent dominate the off-spring, the off-spring is discarded and binary mutation is used to generate a new off-spring. If non of them dominate the other, they are compared to an archive that holds the best solution so far.

Rudolph [Rudo01] introduced a simple MOEA based on a systematic comparison of individuals from parent and of spring population. A non-dominant set is generated from both populations as a result of this comparison. This set become the parent set in the next generation. This algorithm is introduced without simulation to be compared with other MOEAs.

In summary SPEA-II and NSGA-II are the most advanced and widely used MOEA. They efficiently solve the AE problem provided a good evaluation scheme is used.

2.6 Summary

This chapter introduced necessary background on reconfigurable devices, optimization techniques and architecture exploration. The background material in this chapter can be summarized as following:

- Modern FPGAs contain resources and capabilities that enable implementing a complete system on a single chip. Several embedded modules are integrated with the chip fabric beside several cores provided by the vendors to support different DSP blocks.

- Architecture exploration is the task of optimizing system parameters to meet different design objectives. Multi-objective optimization techniques can be used to search the design space of the given problem. Due to the large design space, tools to assist the design are becoming necessary.

- Meta-Heuristic techniques give pareto-optimal solutions within a reasonable time. Different meta-heuristic techniques can be used as search algorithms including evolutionary algorithms that are efficient in exploring the solution space effectively.

Chapter 3

Literature Review

Architecture exploration of embedded systems involves assembling different computation sub-system (processors, custom logic, memories, and peripherals) and communication sub-systems (buses and direct links) to meet the application constraints in the form of speed, area and power consumption. The goal of architecture exploration is to effectively search the design space of a given application to find an optimal or near optimal hardware implementation that meets the specification. This process includes two main steps: searching the design space to generate different possible architectures, and the evaluation of all or a portion of the generated architectures. Depending on the methodology used to perform the exploration, the number of architectures generated will vary, and the number of evaluated architectures will be different. Also, the abstraction level used to perform different tasks is an issue in this process. The different research issues in architecture exploration are illustrated in Figure 3.1. The root of the tree is the architecture exploration process. Different research issues are shown in the second level of the tree. The subsequent levels of the tree show the different methods and approaches used.

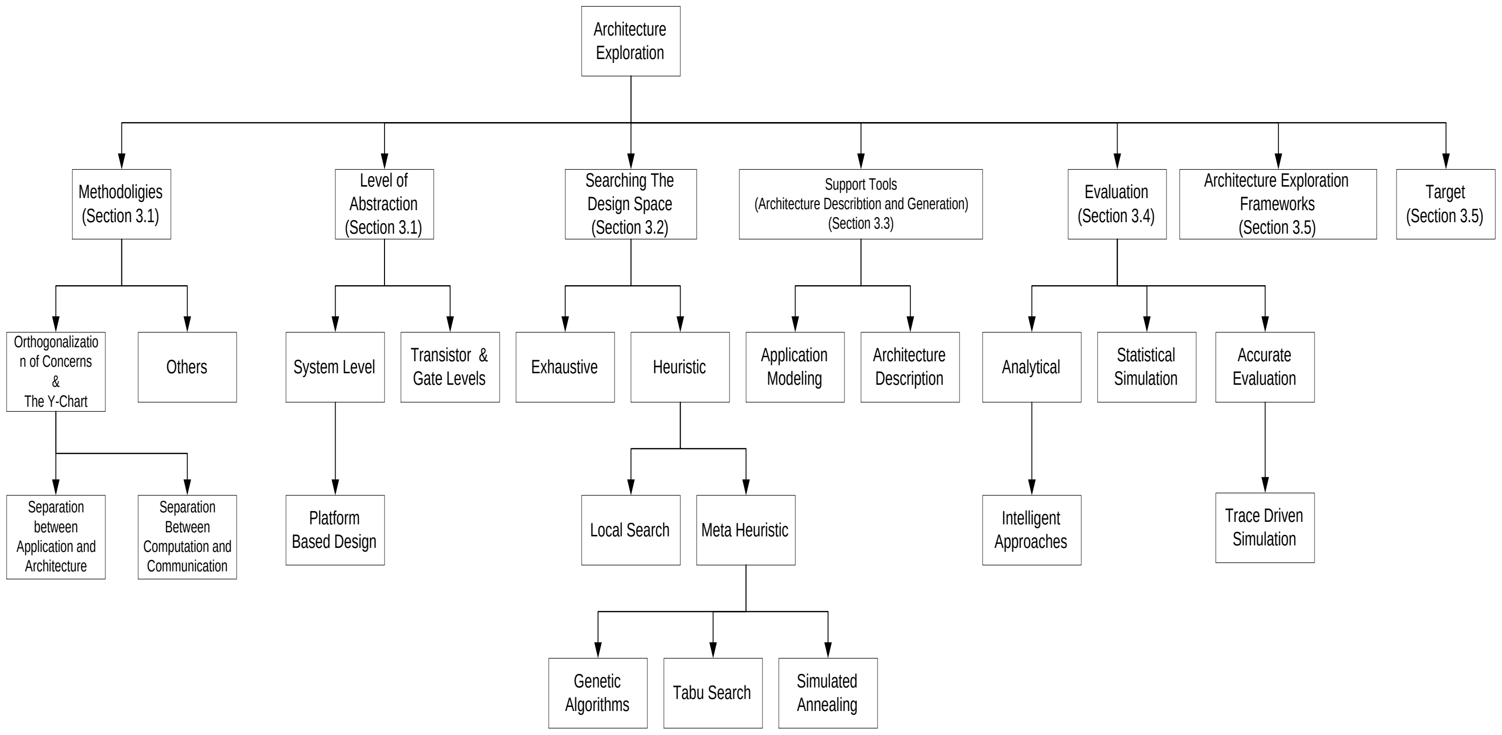


Figure 3.1: Architecture Exploration

This chapter introduces literature review on architecture exploration methodologies, frameworks and different evaluation techniques used to evaluate the quality of solutions obtained. Coarse grained reconfigurable devices are also reviewed as a possible target implementation platform.

3.1 Architecture Exploration Methodologies

Architecture exploration for embedded systems aims at finding the optimal hardware architecture for a given application. This includes the computation resources that will be used in the architecture, the mapping of different application modules into the computation resources, and communication resources used between different components. The exploration methodology defines the modeling method of the application, the level of abstraction used to process the exploration (usually system level) [Grie04], and the goal of the exploration (architecture definition, mapping or communication mapping).

In [Keut00] the concept of orthogonalization of concerns is introduced. This paradigm states that the separation of various aspects of the design allows more effective exploration of alternative solutions. An essential aspect of this design methodology is the separation between:

- function and architecture.
- communication and computation.

The first point indicates that the design of the architecture could be independent from the application itself. This introduced the concept of “Platform Based Design” [SV04, Mart02] in which a common platform could be used for the implementation of different applications by selecting different components from a core library and choosing different design parameters. In this case the role of the exploration tool is to search the design space specified by the platform parameters. The other important task for the tool is to map the application to different components of the platform. The second point enables exploring the communication between the different components in the architecture independently of the architecture itself. By communication we mean bus configurations, topologies and protocols that will be used for data communication between different computation units. Several frameworks are dedicated for the exploration of the communication system of embedded systems [Wang05, Lahi04, Kim06].

The separation between function and architecture is the concept behind the Y-chart scheme discussed earlier in section 2.3.1 and presented in [Liev01]. This scheme permits multiple target applications to be mapped into one candidate architecture for evaluation of their performance. It also enables mapping a single application into different architectures to select the best architecture for a specific application. This scheme is employed in several frameworks and focus mainly on the left side of the chart where the architecture is explored.

In [Giva02b] a methodology is introduced for the architecture exploration of a parameterized platform. To reduce the search time, a graph is built to reflect the dependency between different parameters. The Y-chart strategy is employed to separate application design from the architecture. A mapping stage is used to map the application to hardware. Besides, in this methodology the design space is partitioned into clusters according to dependency between system parameters. Each cluster is exhaustively searched using an iterative approach to find the optimal solution within the cluster. The authors suggest the use of heuristic methods, such as genetic algorithms, to replace the exhaustive search since it is slow. This methodology is used mainly to optimize a parameterized SoC, where the system is composed of a fixed number of parameterized components. The use of this methodology with general architectures, where the computation units are not defined, is not appropriate. That is because the design space in this case is very huge (the architecture is not defined yet) and the design parameters are not will defined [Giva02b, Asci05b].

The separation between the different design aspects of embedded system is a very useful approach as it enables breaking the exploration process into sub tasks that could be investigated separately (investigating the architecture independently from the application for example).

3.2 Techniques for Searching the Design Space

Searching the design space is a basic task that has to be performed efficiently in architecture exploration. In this section a review of different search strategies used in architecture exploration frameworks is given. Frameworks are classified in this section according to the employed search approach used to cover different points in the design space.

3.2.1 Searching the Design Space using Exhaustive Methods

Exhaustive search techniques are used to search the design space by covering all possible solutions in the design space at the expense of high computation time [Forn02].

Exhaustive search is used in [Bech03] for the exploration of an SoC system designed for GPS. The design flow in this work consists of two phases. Exhaustive search is used in the first phase in which coarse-grain exploration is used to investigate different solutions at a high level of abstraction, which shortens the time required to search the design space. In this phase the possible architectures suitable for the given problem are evaluated and one candidate is selected. The selected architecture is tuned in the second phase in which fine-grain exploration is performed to select the best parameters for the selected architecture. Fine-grain exploration is performed at a lower level of abstraction and different optimizations techniques could be used in this phase.

In the Platune framework [Giva02a] exhaustive search is used to search design space clusters separately. These clusters are created by investigating the different design parameters and their dependency. Searching each cluster exhaustively take a much shorter time than exploring the complete design space and all clusters could be searched in parallel.

In exhaustive search, the design space is searched iteratively to find the optimal solution (global minima). As this search technique is a very time consuming, some heuristic should be used. In this section two approaches to speed up exhaustive search are reviewed:

- Perform exhaustive search in a high level of abstraction to speed the search process. Low level tuning is performed on the selected architectures and exhaustive search could be used here to search the design space which is reduced significantly [Bech03].

- Use dependency between system parameters to partition the design space into clusters, which are searched independently using exhaustive search [Giva02a].

In the next few subsections several heuristic methods and advanced meta-heuristics will be further discussed.

3.2.2 Searching the Design Space using Heuristic Methods

Heuristics are used when the exhaustive search is not applicable. Due to the huge computational effort required by exhaustive search, several heuristic methods are used in architecture exploration to speed up the search process.

3.2.2.1 Searching the Design Space using Local Search

Local search heuristic algorithms move from solution to solution in the design space of the given problem until a sub-optimal solution is found or a time bound is elapsed. Local search has the advantage of short search time, with the disadvantage of the possibility of falling into local minima. It is usually combined with pre-analysis of the design space with the hope that will increase the chance of finding the global minima.

Local search is used in [Forn02] to explore the memory hierarchy for embedded systems. The authors introduce an iterative local search algorithm based on sensitivity analysis of the objective function to design parameters. This sensitivity analysis is used to move the starting point of the search close to a suspected global minima. The sensitivity analysis could be used with other search techniques to improve its performance [Giva02b].

Local iterative search is used in [Kim06] to explore bus-based SoC architectures . Limiting the architecture to bus-based SoC reduce the size of the design space which increases the chance of finding the global minimum (which is not guaranteed). The iterative approach used in this work is simple but might get stuck in a local minima. This method is also applied for specific class of architectures (bus based architectures) and can be time consuming when used with more complicated architectures [Grie04]. Local

search is used to build a tool for architecture exploration of dynamically reconfigurable devices and is introduced in [Mira05]. Preprocessing is used to select (i)the starting point of search and (ii)the method used to move from one solution to the other in order to reach a solution at most a few percent away from the global optimum. The accuracy of the final result depends on the choice of the starting point.
In summary, when local search used in architecture exploration, it is usually combined with other techniques. These techniques aim to start the search from a point close to the global minima and reduce the size of the design space to increase the chance that local search find it. These approaches can be summarized in the following points:

- Sensitivity analysis that measures the change of the objective function to every design parameter. This help to reshape the design space to improve the performance of local search [Forn02].

- Reducing the size of the design space by limiting the search to a specific architecture [Kim06].

- Select the starting point and the method used to move from one point in the design space to the other to reach a solution at most a few percent away from the global optimum [Mira05].

3.2.2.2 Searching the Design Space using Meta-Heuristic Methods

Meta-heuristic methods are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic; or when it is not practical to implement such a method. Most commonly used meta-heuristics are targeted to combinatorial optimization problems, but of course can handle any other sort of problems. The goal of combinatorial optimization is to find a discrete mathematical object (such as a bit string or permutation) that maximizes (or minimizes) an arbitrary function specified by the user of the meta-heuristic. As shown in Section 2.3, architecture exploration can be viewed as a form of combinatorial problem. For this reason using meta-heuristic search techniques in architecture exploration attracted several researchers. The use of evolutionary algorithms in architecture exploration is introduced next followed by the use of other meta-heuristic methods.

3.2.2.3 Searching the Design Space using Evolutionary Algorithms

In Chapter 2 the architecture exploration problem was introduced and shown to be a multi-objective optimization problem. For this reason multi-objective evolutionary algorithms introduced in section 2.5.3 are commonly used in architecture exploration [Grie04].

Several approaches are used in the literature to treat multiple objectives for the use in evolutionary algorithms. One common approach is to use Pareto curves to define the range of optimal solutions. From this category there exist several algorithms such as SPEA , and NSGA (more information was given in section 2.5.3).

In [Erba06] the authors give a comparative study between two commonly used multi-objective evolutionary algorithms, SPEA2 and NSGA-II (introduced in Section 2.5.3). The comparison shows that NSGA-II superior SPEA2 in most of the test benches used. They gave almost the same results compared to the Pareto-optimal solution, with NSGA-II is two times faster. SPEA has the advantage of a simpler computation requirement.

SPEA [Zitz99] algorithm and its improvements (SPEA2, SPEA2+) [Zitz01, Zitz02] are used in Spade framework [Liev01], Sesame framework [Pime06] for searching the design space of a heterogenous embedded systems.

In [Asci05a] SPEA2 is used for the exploration of two parameterized architectures; VLIW and RISC. Sensitivity analysis of the system parameters is used to build a dependency graph before searching the design space. This work was an extension of the work introduced in [Giva02a].

The system level architecture exploration framework introduced in [Haub05] makes use of PISA search strategy. PISA uses SPEA in the optimization phase [Bleu03]. This framework is design for the architecture exploration of embedded systems targeting reconfigurable devices (FPGA).

NSGA-II usage is investigated in [Pime06] in heterogenous embedded systems, but the authors selected SPEA for their implementation. NSGA-II is used in [Mouh06] to optimize the implementation of a four processor system on reconfigurable architecture.

SPEA is widely used for architecture exploration although other algorithms such as NSGA (and its variations) out perform SPEA. That is because the simple computation requirement for fitness assignment [Erba06].

Another approach for working with multi-objective without the use of Pareto-curves is to aggregate different objectives into a single objective (cost) function before the actual search [Grie04].

This approach is used in [Shin04] to search the design for SoC design. The framework is oriented toward the optimization of communication system as the backbone of the system. The authors made use of weighted-sum function to combine multiple objectives into a single cost function which is optimized using a simple evolutionary algorithm. The problem of this approach is the selection of the weigh values of each objective. It mainly depends on the design requirements which differ from one application to the other.

The same approach is used in [Kris06] where a graded penalty function is used as an objective function in evaluating the quality of the designs. This framework is design for high level synthesis of data-paths for data-dominated applications. An evolutionary algorithms is proposed in this work based on a multi-chromosome representation to encode the data-path, schedules and module allocations.

Different frameworks make use of multi-objective evolutionary algorithms to search the design space during architecture exploration. The main advantage of evolutionary algorithms is that they cover a wide range of solutions within the design space in a reasonable time. Two common multi-objective evolutionary algorithms based on the concept of Pareto optimality are used for searching the design space: SPEA (and its alternatives), and NSGA (and its alternatives). SPEA has the advantage of a simpler computation model, while NSGA has the advantage of better performance [Erba06].

Multiple objectives could be aggregated into a single cost function that is then optimized using evolutionary algorithms (other optimization or search techniques could be used). A common approach is to use weighted-sum function for the aggregation. While this approach could minimize the computation cost required by the multi-objective version of evolutionary algorithms (SPEA and NSGA), the selection of the weight of each objective is very critical in the quality of the final solution [Grie04].

3.2.2.4 Searching the Design Space Using Other Meta-Heuristic Methods

In this section a review of other meta-heuristic methods in searching the design space during architecture exploration is introduced.

In [Pale03], a framework for architecture exploration of embedded system is introduced. A selection of heuristic methods to approximate a Pareto-optimal curves are used.

The list of methods includes Random Search Pareto (RSP), the Pareto Simulated Annealing (PSA) [Suma04, Suma02], and the Pareto Reactive Tabu Search (PRTS)[Batt94]. These methods are used to explore a parameterized Design Space (DS) that defines a set of feasible architectures. In this work different meta-heuristic methods are combined to perform architecture exploration of a processor based parameterized system to optimize the system parameters. Combining different methods makes use of the benefits of each technique to cover a wide portion of the design space, and overcome the shortage of each method working alone. Performing different search procedures at the same time can be time consuming and reduces the benefits of using meta-heuristics search.

A multi-objective tabu search algorithm is introduced in [Slom04] to be used for the architecture exploration of embedded systems. The introduced tabu search algorithm uses the concept of Pareto optimality to evaluate each solution according to the different objectives. The authors show the results for solving multi-objective optimization in general.

The majority of the research in using meta-heuristic algorithms in architecture exploration is devoted for the use of evolutionary algorithms. That is because of its ability to cover more solutions which increases the chance to reach a global optimal solution.

3.3 Architecture Exploration Support Tools

In this section a review of some tools that could be used in modeling the application and architecture to support architecture exploration is presented.

3.3.1 Application Modeling

Several approaches exist in the literature to model the application at different levels of abstraction. Kahn Process Networks (KNP) [Kahn74] in which concurrent process communication is performed through FIFO-organized, unbounded, unidirectional point to point channels, are used to model applications at a high level of abstraction. Each process represents a single computational task performed on its local data. Read operations from the channel is blocking, the process stop until the required data are available, while write operations are non-blocking because the number of channels is assumed unlimited. KNPs are used in Artemis [Pime01], Spade [Liev01], and Sesame [Pime06, Erba06, Erba03] frameworks to model the workload of the application to be used during exploration.

Symbolic programs yet another approach used to model the application for architecture exploration [Zivk03]. In this method the application is converted into a symbolic program that contains information about the application execution requirements such as loops, conditions, and memory access. This symbolic program is then used during exploration to model the application. As this symbolic language is just an abstraction of the actual application, its evaluation or simulation is much simpler and gives reasonable results. In [Zivk03] an approach is presented to obtain a symbolic program for a given application.

In [Govi05] a petri-net model is used for Network processor architectures. This model is used with the Intel IXP architecture [Corp03] to provide a simulation model that can be used to evaluate different candidate architectures. This work is used with a specific architecture but the idea of petri-net modeling can be used to model applications for the architecture exploration tool.

Ptolemy is a Java-based component assembly framework with a graphical user interface called Vergil [Le03]. The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interactions between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation. Ptolemy includes a growing suite of domains, each of which realizes a model of computation. Examples of these models Continuous-time Modeling (CT), Dynamic Dataflow (DDF) ,Discrete-event Modeling (DM), Finite State Machines and modal model(FSM), Process Networks (PN), and Synchronous Dataflow (SDF). The framework also includes a component library, in which most components can operate in several of the domains. Ptolemy and its new version Ptolemy-II provide a good environment for the modeling application in different computational models, and it can easily be integrated with other tools [Le03].

3.3.2 Architecture Description

The architecture of an embedded system could be described at different levels of abstractions during architecture exploration. Different tools and frameworks introduce different approaches for architecture description.

The PIRATE framework [Pale04] is used for exploration of a parameterized multiprocessor SOC architecture. In this architecture, IP Cores are connected using parameterized interconnection elements and switches. The framework generates an RTL description of the candidate architecture. This allows power estimation and performance evaluation using Synopsys tools. The authors introduce an RTL generator for different architectures depending on the given user specification. This can be used for automatic architecture exploration but the authors do not introduce one in their work. Automatic generation of RTL models for different architectures could be employed in the architecture exploration process, as it enables evaluating the generated architecture at the RTL level. However, since evaluation at the RTL level might be time consuming, moving to higher levels of abstraction is recommended. The V-SAT tool is introduced in [Khar01] which is a visual tool for SoC exploration. It has three main components, EXPRESSION, which is an Architecture Description Language (ADL), SIMPRESS, a simulator for architecture analysis and evaluation, and V-SAT GUI, user interface for easy specifications and detailed analysis. The tool does not include an automatic exploration feature. The architecture description language and the simulator can be used for early design stage manual/automatic exploration.

In [Mish03] the EXPRESSION language is used for exploration of pipelined processors by automatically generating an RTL model for the candidate architecture. The generator tool generates a synthesizable RTL from ADL specification according to the user constraints. The authors developed a VHDL model for each generic function and subfunction that can be used in the implementation of the given architecture. Following synthesis the architecture is evaluated and feedback is sent for further modifications. Using RTL description allows accurate simulation at RTL and Gate levels. But this will be very time consuming. Also the tool does not employ the use of soft-core processor as an implementation component, which affects the flexibility of the resulting system.

3.4 Evaluation Techniques for Architecture Exploration

The evaluation of embedded systems plays an important role in the architecture exploration process. Every candidate architecture generated during the search process should be evaluated to measure its optimality. Accurate evaluation should guide the search process towards a near optimal solution. Another effect of the evaluation process is that accurate simulation is usually time consuming (gate level, cycle accurate simulation produce an accurate evaluation in the cost of long simulation time). Evaluation at a higher level of abstraction can reduce the evaluation time significantly with the cost of reducing the level of accuracy.

Several articles dealing with the evaluation of embedded architectures exist and we attempt to review and summarize them in the following subsections. First accurate simulation is covered, followed by statistical simulation and analytical evaluation.

3.4.1 Accurate Simulation

In accurate simulation the behavior of the system is modeled on a clock cycle basis. It means at any instance of time the state of the simulated model is identical to the actual implementation. This enables extracting accurate performance measurements before implementing the design. Accurate simulation could be performed at different levels of abstractions. Instruction set simulators are required to simulate the behavior of different processors at the instruction level. While the same processor could be simulated at the RTL level while integrated with other components of the design. In this section a review of some tools that is used to perform accurate simulation is given.

The WARTS framework is introduced in [Hill93]. It provides a set of tools for (i) profiling applications running on MIPS and SPARC systems, (ii) cache performance profiler. (iii) Cache simulator. This tool set can be used to analyze application performance on a specific architecture. However, it does not provide any power analysis.

SIMIC tool is introduced in [Magn95]. It is an instruction level simulator. It can be used for efficient memory system analysis in embedded systems. Again this framework does not support power analysis. It can only give performance estimation for the memory system, hence can help in system optimization.

SimpleScalar [Burg97] is yet another tool set for simulation of MIPS like architectures (SimpleScalar architecture). This tool set provides a fast and accurate simulation for applications running on an architecture derived from MIPS-IV ISA. The authors provide a set of tools for functional simulation, cache simulation, profiling and timing simulation. However the tool lacks any power analysis as well.

Energy estimation and optimization is added to SimpleScalar using SimplePower framework introduced in [Vija00]. SimplePower makes use of the SimpleScaler tool-set to perform a transition sensitive, cycle accurate energy evaluation.

WATTCH framework is introduced in [Broo00]. It is built over SimpleScalar and provides it with a power evaluation methodology that is missing in SimpleScalar. The WATTCH framework is widely used for evaluation of candidate architectures in architecture exploration operation.

An Instruction Level Simulator (ILS) is introduced in [Hadj03, Hadj99]. This simulator provides an accurate performance evaluation for different architectures specially VLIW architectures. This tool can be used for efficient architecture exploration. The exploration tool can generate different architectures using the Instruction Set Description Language (ISDL) [Hadj00]. This language is used to describe different architectures.

After describing the architecture using ISDL, the ILS is used to performance evaluation of the new architecture. The tool set also includes a synthesis tool from ISDL. It can help in later development stages.

The High level Performance Estimator (HiPerE) is introduced in [Moha02]. This tool is part of the MILAN framework [Baks01] which is an integrated framework for simulation of embedded systems. The authors introduce a General Model (GenM) which captures the capabilities of a large class of SoC architectures. This simulator is used for architecture exploration in MILAN framework [Baks01].

Accurate simulators provide cycle by cycle measures of the embedded system. This process is very time consuming and is suitable only for the verification of the final design.

3.4.1.1 Trace-Driven Simulation

In trace-driven simulation, an initial program run is performed to extracts all memory accesses and store them in a trace which is used for performance estimation. It is more efficient in the estimation of the performance of the memory sub-system. But the concept could be applied for the estimation of the complete system performance. Trace driven simulation is used in several frameworks for the evaluation the generated architecture. In [Kim06] it is used for the evaluation of bus-based SoC architecture. In [Bech03] it is used for the evaluation of a multi-processor configurable chip (SPP chip-set). And in [Liev01] it is used for the evaluation of a heterogenous embedded system.

3.4.2 Statistical Simulation

The basic idea in statistical simulation is to model a workload's important performance characteristics with a synthetic trace, and execute the trace in a statistical simulator to obtain a performance estimate. Because the performance estimation quickly converges, the simulation speed is improved dramatically and so it is a good choice for architecture exploration [Josh06]. In [Oski00] Hybrid Processor Simulator (HLS) is proposed. It uses statistical and symbolic execution to evaluate design alternatives. It does not simulate a precise order of program instructions, but it uses statistical profile of the application program to generate a synthetic instruction stream. This stream, is symbolically executed like normal simulators. The statistical profile is based on the workload of the application. The authors use a processor structure close to the one used with SimpleScaler simulator [Burg97]. This structure is configurable to enable validation against MIPS R10K processor.

An improved version, HLS++ is introduced in [Bell04] by modeling the workload at the granularity of the basic block and by changing the processor model to more closely reflect components in modern microprocessors. The authors claim that these modification raises the accuracy of HLS by a factor 3.78.

In [Josh06] an enhanced version of HLS++,(SS-HLS++) is introduced. This tool profiles the input program to collect statistics about its execution (Statistical Profile). The statistical profile is used to generate a synthetic trace. The instructions in the synthetic trace are simulated using trace-driven simulator to obtain a performance estimate. They introduced an improvement in each step to increase the accuracy of HLS++.

Statistical simulators in general are faster, and less accurate than cycle-accurate simulators. They require the existence of statical information about the application. The statistical information requires an efficient profiler which might not be available for most of the processors used in the framework.

3.4.3 Analytical Evaluation

In analytical evaluation the performance of the embedded system is estimated using analytical models that give a high level of abstraction of system components. The overall performance of the system is estimated using the component models. This estimation is performed using analytical evaluation scheme that is developed from the knowledge about the behavior of the system. Evaluation results are obtained in a reasonable time compared to accurate simulation. The accuracy of analytical evaluation depends on the accuracy of the developed analytical model for system behavior and components.

A combined simulation and analytical estimation framework is introduced in [Chak03b] for network processor architectures. The authors propose an analytical model for network processors that can be used for performance simulation and evaluation for such systems. Their analytical model is based on a real time calculus introduced in [Chak03a] that can be used in the analysis of various system properties, timing and loads of different components.

Analytical evaluation provides an estimation for the performance of embedded systems. The accuracy of analytical evaluation is lower than other evaluation techniques, but the speed gained from using this technique makes it suitable for use in the process of architecture exploration.

Some work is presented for using artificial intelligent approaches such as neural networks and fuzzy logic combined with analytical analysis to improve the its accuracy. Neural networks are used for performance estimation in [Oyam04]. This work introduce a methodology for training and using neural networks for the estimation of the performance of different architectures.

In [Hart01] a survey of different CGRA architectures, and fuzzy logic is used for the evaluation of different implementation during architecture exploration of these architecture. The use of artificial intelligent approach for embedded system evaluation is very interesting and requires more investigation.

In [Yi06] a survey of simulation and benchmark tools available and their use in architecture exploration is introduced. It gives a good comparison of different types of simulators and detailed steps involved in using them for architecture exploration.

3.5 Architecture Exploration Frameworks

Different frameworks are found in the literature that support architecture exploration. A group of these frameworks are geared for the exploration of a specific parameterized architecture. A parameterized architecture is a predefined architecture that has fixed main parameterized computation units, and it has the ability to change parameters such as memory size, bus width, cache size and cache association [Giva02b, Giva02a]. The architecture exploration framework goal is to choose system parameters that meet the application requirements. Another group of frameworks deal with a generalized architecture model (i.e., no predefined architecture). The framework is responsible for generating a suitable architecture, including all the system components and parameters for the given application. These frameworks may also include tools for profiling and simulation. Some frameworks are limited for the exploration of the memory hierarchy of the architecture. Other frameworks are dedicated for exploring the communication subsystem of embedded systems. In this case the computation sub-system and the mapping of different software blocks to computation sub-system is defined and the framework is used to optimize the communication between different computation modules according to the software model.

3.5.1 Exploration of the Communication Sub-System

In this section a review of some frameworks that deals with the exploration of the communication sub-system is introduced. Communication sub-system refers to different buses , direct links, and protocols used for the communication between different computation modules in the architecture.

An architecture exploration approach for SoC design is introduced in [Shin04]. This approach is centered on the optimization of the communication sub-system and during this optimization application mapping is performed. The concept of separation between computation and communication is employed here. This work makes use of a standard parameterized bus architecture as the backbone of the system. A simple genetic algorithm is used for the optimization of the parameterized bus system. The optimization is performed in three steps. First, a system interconnect is optimized based on specifications with pseudo masters that characterize the behavioral requirements of them. Then, the mapping of multiple IPs to multiple interconnects is to be optimized. (Hardware/software partitioning is also performed in this phase.) Finally, each IP is fine-tuned to fully exploit the optimized interconnect. The GA (evolutionary algorithm) is used to generate a population of configurations using normal operations (crossover , mutation). Each architecture corresponding to a configuration is simulated at the RTL level using cycle accurate simulation. The simulation results are used to select the best individual. The authors also compare GA with simulated annealing. They suggest that GA outperforms SA according to their experimental results as it finds optimal solution faster and more reliable in general, while SA heavily depends on the quality of the initial configuration. The evaluation of the generated architectures is performed using tools provided by the bus architecture provider. That makes this approach very dependant on the used architecture, although the authors assumes that this approach could be applied for general SoC design.

Iterative exploration for bus-based SoC architecture is introduced in [Kim06]. The authors introduce a multi-step iterative approach for bus architecture exploration. The technique is composed of three phases. In the first phase a list of possible candidates is quickly generated and evaluated using performance estimation method introduced in [Kim03], which has an estimation error of 10%. This estimation method is based on the queuing model of the system where processing elements are regarded as customers and a bus with its associated memory is regarded as a single server. The performance is estimated by calculating the utilization of that server. In the second phase of the exploration process trace-driven simulation is used to carefully examine the candidate architectures selected earlier. If the performance of the best candidate is not improved from the previous iteration, the search loop is terminated. Otherwise it continues to phase three, in which a new set of candidate architectures are generated by changing the processing elements or the bus architecture and is fed to the next iteration. The iterative approach used in this work is simple but might get stuck in a local minima. This methodology also is applied for specific class of architectures (bus based architectures) and can be time consuming when used with more complicated architectures.

Frameworks in this category make use of the concept of separation of concerns between computation and communication sub-systems. The optimization of the communication sub-system is performed separately from the computation system.

3.5.2 Exploration of the Computation Sub-system

Computation sub-system refers to the different components of the systems such as processors, memories, peripherals and hardware accelerators. In this section different frameworks used for exploring the computation sub-system is presented.

3.5.2.1 Frameworks for the Exploration of Parameterized Architecture

This category of frameworks deal with the exploration of parameterized SoC, where the system is composed of a fixed number of parameterized components. The framework is used to optimize these system parameters.

A framework for architecture exploration of embedded system is introduced in [Pale03]. A selection of heuristic methods to approximate a Pareto-optimal curves are used. These methods are used to optimize a parameterized system. The WATTCH frame work [Broo00] is used for simulation and evaluation for the different resulting architectures. The framework is used to optimize a super-scaler microprocessor-based system.

A framework for architecture exploration for an SoC system designed for GPS hand held devices is introduced in [Bech03]. The framework is used to explore the design space of a multi-processor configurable chip (SPP chip-set). The design flow consists of two phases. The first phase performs a coarse-grain exploration in which the solution is investigated exhaustively at a higher level of abstraction. In this phase the possible architectures suitable for the given problem are evaluated and one candidate is selected. Using high level of abstraction during this phase shortens the time required to search the design space. Two performance metrics are used for system evaluation; the execution time and power consumption. The selected architecture is fine tuned in the second phase in which fine-grain exploration is performed to select the best parameters for the selected architecture. Trace driven simulation is used for evaluating the candidate architectures. The application is modeled using a graph that defines the work load of the application on the different architecture components. Using two-steps optimization is supposed to reduce the search time for the optimal solution. The main idea of this work is to use a high level of abstraction during the first phase that performs the real search. This approach does not guarantee that the selected architecture is the optimal one. It might give satisfying results for the chip-set and the specific applications it designed for.

The architecture exploration methodology introduced in [Giva02b] is used to build the Platune framework [Giva02a] which is also based on a parameterized multi-processor SoC introduced in [Giva02b]. The framework includes a set of simulators for the CPU, cache and memory for architecture evaluation. As discussed before in section 3.1 the methodology is based on finding the dependency between the parameters of the SoC. So applying the methodology on other architectures my not be possible. Also when the architecture is not defined the task will be more difficult. In [Asci04a] multi-objective evolutionary algorithms replaced exhaustive search for exploring the design space clusters generated by the dependency graph.

Another framework based on the multi-objective evolutionary algorithms (MOEA) is introduced in [Asci05a]. SPEA2 is used to search the design space. The authors make use of SPEA2 for the exploration of two parameterized architectures, VLIW based architecture and RISC based architecture introduced in [Giva02a]. Their VLIW architecture is described in [Asci01, Asci05b]. The use of SPEA2 in exploration is discussed in [Asci04a, Asci04b]. The authors show a detailed description of using SPEA2 to model their parameterized architecture. The concept of dependency graph that enables clustering the design space according to the parameters of the underlying architecture [Giva02b], is used with the VLIW architecture. Instead of using exhaustive search, MOEA is used to search each cluster. The dependency graph will remain the issue of this approach as it is architecture dependent, and with more generalized architecture it might be impossible to form such a graph.

Design space exploration for a multi-processor on chip is introduced in [Mouh06]. In this work NSGA-II multi-objective evolutionary algorithm is used to optimize the implementation of a 4-processor system on FPGA. MicroBlaze soft-core processor is used to build the multi-processing system. The system is used to implement a simple network on chip protocol for message transfer between the four processors. The goal of this framework is to optimize the implementation resources of the FPGA. Three objectives are selected for optimization: number of used blocks of the block ram (memory blocks on FPGA), number of slices used, and number of cycles used. By varying the configuration parameters of four MicroBlaze soft cores, different alternative architectures could be generated. The evaluation of each architecture is performed through the real implementation on FPGA board connected to a host workstation. The information collected from the real implementation is used to measure the optimality of each generated architecture. The authors compare this evaluation approach with cycle accurate RTL simulation of each generated architecture, and their results show that the physical implementation gives faster evaluation than RTL simulation, although it remains in the hours range. This approach is suitable only for implementing multi-processing systems using soft-core processors on FPGA. It can not be applied for hard core processor, where no configuration could be made to control resources, or with ASIC implementation where real implementation is not possible.

3.5.2.2 Frameworks for the Exploration of Generalized Architectures

In this section a review of architecture exploration frameworks used for the exploration of a generalized architecture is given. These frameworks are used to generate a sub-optimal heterogenous architecture for a given application.

An architecture exploration tool for dynamically reconfigurable architectures is introduced in [Mira05]. A local search algorithm is used to build a tool that aims to select computational resources, mapping of application nodes to the computational resources, and task schedules on the programmable processors for the implementation on a dynamically configurable device. Two performance metrics are used during optimization: computation time, and solution cost. This tool is designed specifically for configurable devices (FPGAs) that support dynamic configuration (an example is Virtex family from Xilinx). The selection of the starting point of search and the move from one solution to the other are developed in order to reach a solution at most a few percent away from the global optimum. The disadvantage of this tool is that it is designed to employ a specific task graph that represents the scheduling of the application and the mapping of each node to the hardware architecture. This graph is used to evaluate each solution during the search by determining the longest path in the graph, which give an approximate evaluation of the systems. Combined with the local search algorithm used the resulting solution will be far from the global optimal solution.

The Architectures and Methods for Embedded Media Systems (Artemis) [Pime01] is a design framework that provides tools of modeling and simulation for embedded systems. It efficiently explores the design space of heterogeneous embedded systems, at multiple levels of abstraction, and for a wide range of applications that run on these architectures. The Artemis framework is based on two simulation frame works: Spade framework (system-level performance analysis and design space exploration)[Liev01], and the Sesame framework (simulation of embedded-system architectures for multilevel exploration)[Pime06, Erba06, Erba03].

The Spade framework is introduced in [Liev01]. It is used with heterogenous signal processing systems to quickly build models of architectures at a high level of abstraction. The application is modeled using Kahn Process Networks [Kahn74] in which processes are connected through unbounded FIFOs. This framework is integrated with the exploration system in [vdH00] to generate and evaluate different architectures in the design space. The evaluation is performed through Co-Simulation of the application and the architecture using trace driven simulation.

The Sesame framework is introduced in [Pime06]. In this framework multi-objective optimization evolutionary algorithms (MOEA) are used for architecture exploration. The architecture exploration problem is formulated in [Erba03]. Their evaluation model and the use of MOEA is introduced in [Erba06]. This framework includes a complete set of tools that can be used for the architecture exploration of heterogenous systems.

A system level architecture exploration framework based on evolutionary algorithms and slack-based list scheduler is introduced in [Haub05]. This framework supports explicit communication modeling and time-multiplexed architecture modeling in a single model. The application is modeled using a process graph. This framework employs PISA [Bleu03] evolutionary framework as the optimization engine. The authors emphasis the support of explicit communication during system level design space exploration. This framework is used for the architecture exploration of embedded systems targeting reconfigurable devices (FPGA), when multiple configurations are required to be scheduled on a single device. The exploration process handles both the scheduling and communication optimization. Architectures are modeled using a process graph where both scheduling and communication could be represented.

In a system level design methodology is introduced for the exploration of the memory hierarchy for embedded systems. Their goal is to specify the cache size, memory size, association levels and other parameters that define the memory sub-system. To reduce the search time, the authors introduce an iterative local search algorithm based on the sensitivity analysis of the cost function with respect to the tuning parameters of the memory sub-system. Sensitivity analysis entails measuring the change of the objective function to every design parameter which is performed in the first phase of the optimization process. In the second phase a local search algorithm that exploits the sensitivity information is performed. This approach is assumed to perform a faster search. Evaluation of the possible architectures is performed using a mix of cycle-accurate Instruction Set Simulator (ISS), and analytical evaluation. Sensitivity analysis here is used with local search which limited the benefits of this method. Using other search techniques with the sensitivity analysis could result in avoiding local minima.

A framework for design space exploration during high-level synthesis of data-paths for data-dominated applications is introduced in [Kris06]. The framework uses GA to concurrently perform scheduling and allocation with the aim of finding schedules and module combinations that lead to superior designs while considering user-specified latency and area constraints. The authors propose a new GA technique that makes use of a multi-chromosome representation to encode data-path schedules and module allocations and efficient heuristics to minimize functional and storage area costs, while minimizing circuit latencies. The framework provides the flexibility to perform different types of scheduling using a simple and fast list-scheduling technique. A graded penalty function is used as an objective function in evaluating the quality of designs to enable the GA to quickly reach areas of the search space where designs meeting user constraints are most likely to be found. Using GA gives the framework the ability of generating several alternative data-path designs. The proposed GA also performs register minimization for the data-path design. The proposed GA has the advantage of using multi-chromosome to represent different architectures,but it is designed for a specific problem. Applying this approach for other applications requires further investigation.

Different frameworks discussed in this section are summarized in Table 3.1. Several architecture exploration frameworks deal with the exploration of a parameterized SOC platform based on a single processor by changing the platform parameters (Memory size, Cache size and communication configuration). Other frameworks deal with the exploration of a generalized architecture which may include several processors communicating through a communication network. Mutli-objective evolutionary algorithms are commonly used for searching the design space during architecture exploration. Different evaluation techniques at different levels of abstractions are used for the evaluation of the generated architectures.

Project	Year	Architecture	Modeling	Optimization Technique	Employ Y Chart	Abstract Level	Evaluation
[Kim06]	2003	Communication Subsystem (Parameterized Bus Architecture)	RTL Model	GA	No	RTL Level	Cycle Accurate Simulation of the RTL model.
[Shin04]	2004	Communication Subsystem	Queuing Model	Multi-Step Iterative Approach	Yes		1st Phase: Queuing Model 2nd Phase: Trace Driven Simulation.
Platune [Giva02a, Giva02b]	2001	Computation Sub-System (Parameterized SOC based on MIPS R3000)	Parameterized SOC	Cluster Design Space using Parameter Dependency Graph. Exhaustive search explores each cluster.	Yes	System level	Different Simulators for CPU, cache and interconnection buses.
DSE [Pale03]	2003	Computation Sub-System (Parameterized SOC)		A Collection of Heuristic Methods (RSP, PSA, PRTS)	No	System level	Using WATTCH Framework [Broo00]
[Bech03]	2003	Computation Sub-System (Parameterized Multi-processor Chip)		Multi-Level Exhaustive Exploration	No	Multi-Level	Trace Driven Simulation
Ascia et al [Asci05a, Asci04a, Asci05b, Asci04b, Asci01]	2004-2005	Computation Sub-System (Parameterized VLIW Architecture)	Parameterized SOC	Cluster Design Space using Parameter Dependency Graph. SPEA2 explores each cluster.	Yes	System Level	Estimation of Different Measures (Performance, area, and power)
[Mouh06]	2006	Computation Sub-System (Parameterized Multi-processor System Using MicroBlaze)	RTL Model	NSGA-II Multi-Objective Evolutionary Algorithm	No	RTL Level	Performance Measured from Physical Implementation

Table 3.1: Summary of Architecture Exploration Frameworks

Project	Year	Architecture	Modeling	Optimization Technique	Employ Y Chart	Abstract Level	Evaluation
SPADE [Liev01]	1999	Computation Sub-System (Explore Heterogeneous Signal Processing Systems)	Kahn Process Networks	Integrate with the framework of [vdH00] to generate and evaluate different architectures in the design space	Yes	System Level	Co-Simulation using Trace Driven Simulation.
[Forn02]	2002	Computation Sub-System (Memory Hierarchy Exploration)		Sensitivity Analysis and Local Search	No		Cycle Accurate Instruction Set Simulator (ISS)
Sesame [Pime06, Erba06, Erba03]	2006	Computation Sub-System (General Heterogeneous Architecture)	Kahn Process Networks	Multi-objective Optimization Using SPEA2	Yes	System Level	Co-Simulation using Trace Driven Simulation.
[Mira05]	2005	Computation Sub-System (Dynamically Reconfigurable Architectures)	Task Graph and RTL Model	Local Search	No	RTL Level	Cycle Accurate Simulation
[Haub05]	2005	Computation Sub-System (Embedded Systems Targeting Reconfigurable Devices)	Process Graph and RTL Model	PISA Evolutionary Framework	No	RTL Level	
[Kris06]	2006	Computation Sub-System (Scheduling and Allocation for Data-Path Design)		GA with graded objective function, and Multi-Chromosome Representation	No		

Table 3.1: Cont ..

3.6 Target Implementation - CGRA

Architecture exploration of embedded systems aims to propose an efficient hardware implementation for a given application. The target platform of the architecture exploration process affects the design space explored. In section 2.1 a background of the different implementation approaches was given. An important class of these approaches is reconfigurable devices. Reconfigurable devices such as FPGA was the goal of several architecture exploration frameworks in the literature such as [Mira05, Haub05]. Coarse grained reconfigurable arrays are another possible platform for implementing of embedded systems. In this section a review of some coarse-grained reconfigurable arrays (CGRA) and their support for architecture exploration are presented.

Fine-grained reconfigurable logic devices (FPGAs) allow the design of hardware down to the required bit level through configurability. If an application requires 7 or 17-bit arithmetic for an operation, the hardware can directly be configured to fit what is needed. The configurability of this devices comes at the cost of more circuit area, more power consumption, and lower speed. Every level of configurability requires more multiplexing, buffering, routing, and/or memory, thus, requiring more transistors and their interconnection.

Several researchers have studied the use of more coarse-grained reconfigurable devices in [Also00, Bitt97, Chen92, Wang93, Ebel96, Gold00, Hart94, Haus97, LN03, Mars99, Mirs96, Miya98, Srik00, Wain97, Zhan00]. These devices consists of an array of more coarse operators forming a reconfigurable computing machines. This category of devices has the advantages of less circuit area, less power consumption, and higher performance. The use of coarse-grained configurable arrays (CGRAs) also make it easier

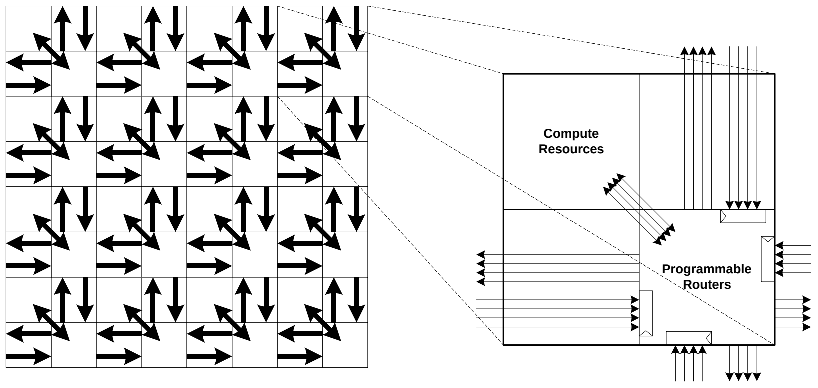


Figure 3.2: Raw Microprocessor Array Architecture

to target for higher level development tools. System level design and platform based design rules can be applied to build tools for CGRAs. In [Hart01] a comparison between 19 different architectures was carried out. The comparison shows that there is no common structure that can be used for all applications, which means that the design of CGRAs is application dependent. Coarse-grained operators inside CGRAs are optimized to work in a specific application domain to gain an efficient implementation. The application should meet the domain requirement of the CGRA to gain a significant improvement compared with other configurable devices.

The Raw chip is introduced in [Tayl02, Wain97] from MIT. It is a two-dimensional array of programmable tiles, each having: a 32-bit MIPS-like microprocessor, local instruction and data caches, and a 32-bit pipelined floating point unit (FPU) as well as several routers and writing channels to support the four on-chip 2-D mesh networks. The structure of this chip is shown in Figure 3.2. It shows a 4×4 Raw array that has actually been fabricated [Tayl02].

The Raw machine uses a switched network for communicating directly between the processors. The length of wires in the architecture is bounded by the width if a tile and each routing segment is registered on tile boundaries. There are two types of communication networks. One is statically configured for predictable performance. This type provides a high performance compared to traditional multi-processor communication. The other type is dynamically configurable through wormhole routing [Bitt97].

The structure of the static network of the raw machine allows it to operate as a pipeline of ALUs and FPUs for a stream of data.

The dynamic networks, on the other hand, are used for less predictable forms of data movements such as cache misses, some forms of data I/O, and operations that happen only occasionally.

Most of the internal operations are handled by the compiler. The compiler control cash coherency, cache misses, and routing of data over the internal routing.

Another example of CGRA is PipeRench [Gold00, Schm02], a project from Carnegie Mellon University. It was implemented with the goal of developing a reconfigurable device for hardware virtualiztion. The structure of this device is shown in Figure 3.3 by employing runtime configuration. The structure of the device enables any application even if it does not fit to still execute. The use of coarse-grained architecture reduces the amount of configuration data that must be quickly swapped in and out of the hardware regions in the CRGA.

The hardware is organized in pipeline stages called “stripes”. Each stripe consists of 16 processing elements (or PEs) that contain 8-bit-wide logic and an 8-entry register file. PEs in each stripe are interconnected in a way that support virtualiztion.

Each PE contains shifters and multiplexors that can be configured to operate on inputs, and a collection of 8 3-bit LUTs. In addition to the LUTs, specialized carry logic is

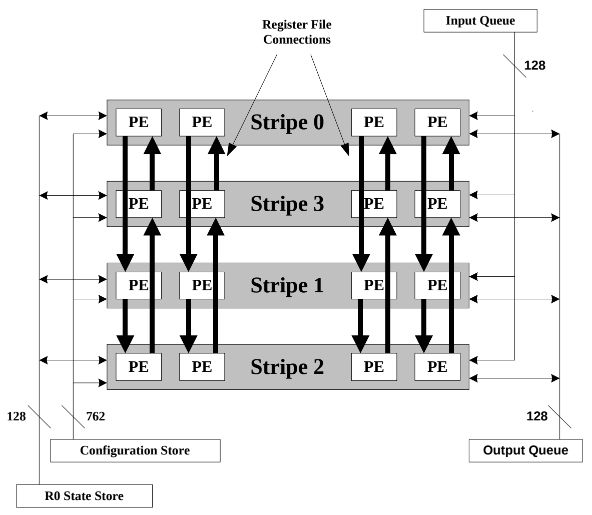


Figure 3.3: PipeRench Architecture

also included to support fast addition. Using the interconnect available, the PEs can be easily combined to form wider operations, including wide shifts using the input shifters. The configuration of the entire stripe require transferring 672 bits of data.

If the application does not fit the physical hardware, the entire stripe can be swapped out and other stripes can be loaded. As the amount of information required to perform swapping is small, this structure can be used for real time virtualization of large application.

On chip registers are used to store the configuration data of each strip, both running and swapped out. So if a strip is required to be swapped out, it is transferred directly from there registers. The structure of PipeRench can store up to 256 virtual strips [Schm02].

PipeRench supports applications with only limited feedback. Despite this, many data-path-oriented application-such as FFTs, DCTs, and many encryption algorithms-require only feed-forward structures and map reasonably well to the architecture. This structure combines the concept of coarse-grained operators, and runtime reconfigurability.

Another CGRA architecture called RaPiD [Ebel96, Cron99, Cron98, Ebel97] (for Reconfigurable Pipelined Data-path) was developed by the University of Washington.

This structured is developed to be a coarse-grained architecture, that can be targeted to a specific application domain and support development at high level of abstraction. RaPiD's architecture provides the end user with the required tools to ease the application development. Other researchers still investigating the same architecture and related ones [Comp04, GmbH].

The structure of RaPiD architecture is shown in Figure 3.4. As shown the coarse-grained functional units has a common data bus with width between 8- of 32-bits. There is a single flexible routing channel connect these channel.

Data stream from external memory is controlled through a Stream Manager. It control the data in/out from the CRGA architecture. This architecture supports runtime reconfiguration. Some functional units can be altered during the operation of the application. The dynamic configuration is controlled through an Instruction Generator, and a Configurable Instruction Decoder.

The number and function of the functional units depends on the application domain for which the device will be used. ALUs, multipliers, registers, and even processors. Other units can be added to depending on the application. For example hardware FFT unit can be added for the DSP application domain.

The routing channel has segments of various lengths to support communications at different distances between function units. This interconnection is also dynamic to support runtime configuration. This routing structure has the ability to support feedback connection through some segments of the routing system. This enables implementing

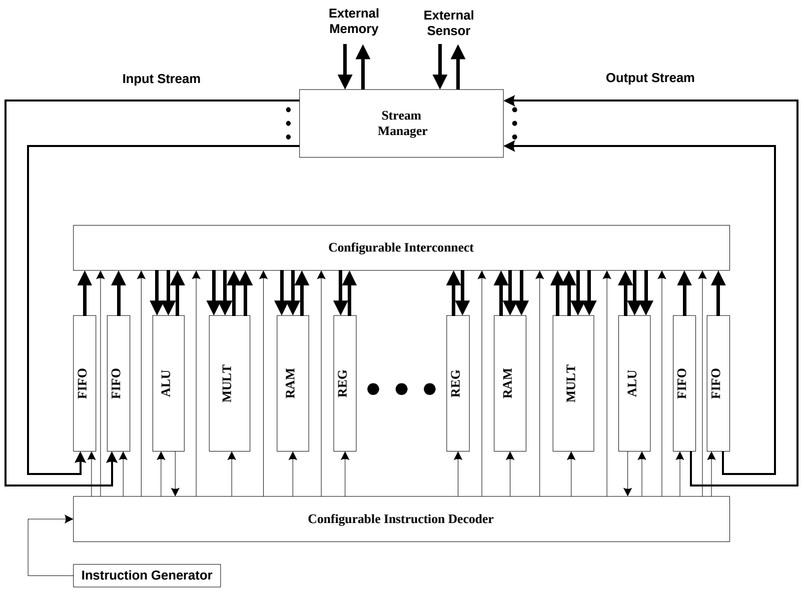


Figure 3.4: RaPiD Architecture

some logic that require feedback.

The Stream Manager, which produces the input data and consumes the output data, is essentially a memory interface with an address generator and FIFO for each input or output stream.

XPP is a commercial CGRA and stands for the eXtreme Processing Platform(XPP) [Baum03, GmbH]. It is a computing array with a data-driven processing model and hierarchial configuration management developed by PACT ”Informations techologie GmbH”. The PACT XPP was developed to handle streaming data applications such as signal or media processing. It is provided as an IP for custom VLSI implementation.

The architecture of the XPP device is shown Figure 3.5 at four different levels. The XPP device consists of several Processing Array Clusters (PACs). A configuration management (CM) unit controls the configurability of the array. There are different level of configurability management. Supervising CM control the overall configurability of the device. Each cluster has a CM that controls its configurability, forming a configuration management hierarchy.

The PAC itself is an array of Processing Array Elements (PAEs) connected through switch boxes for routing between vertical and horizontal busses, and switches for segmenting the horizontal busses. Each PAC also includes I/O resources as well.

Each PAE contains three units: a function unit, a forward register (FREG), and a backward register (BREG). The function unit can be an ALU Object or RAMs. The FREG and BREG objects are used for routing support, data flow, counters, adder/subtractors and barrel shifters.

The ALU object can consume and produce data based on the data driven computation model. It produces two types of packets, data packets that contain the results of an operation and the event packets that contain the condition or state bits resulting from the operation.

XPP makes use of a data-driven computation model which mean that functional units operate only when all of their inputs are available, and their output is read by the next XPP unit. The CM units communicate with PAEs to know when they can be reconfigured. This enables runtime reconfiguration of the PAEs during operation.

The hierarchical configuration management system also enables configuring different parts of the device with different applications. So each part of the device operate independently.

Field Programmable Object Array (FPOA) produced by MathStar [Helg03] is another example of CGRAs. The MathStar architecture is intended to be optimized for a particular application domain. The customer chooses the functional units required for his application. This functional units is called Silicon Objects. These objects are designed to

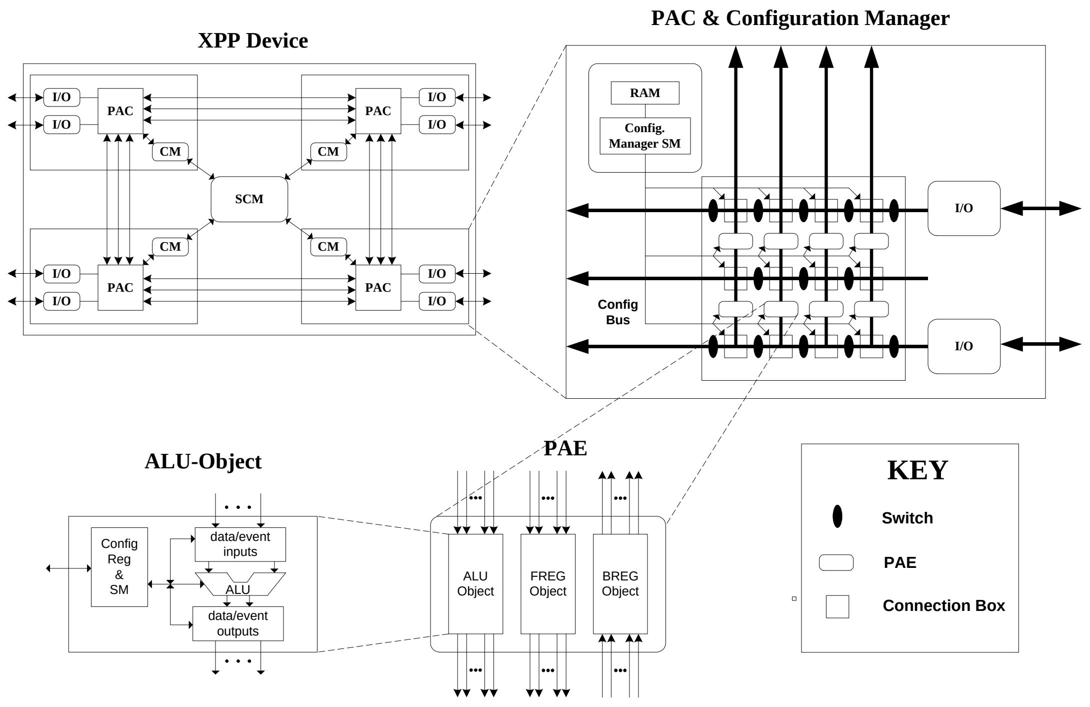


Figure 3.5: PACT's eXtreme Processing Platform

fit in any part of the design. This makes the device cost effective and shortens the time-to-market. The device is structured as a two-dimensional object array. The design of this FPOA provides a time to market of less than a month with 1-GHz operation speeds. Figure 3.6 shows the internal structure of this device. The array can be heterogeneous or homogeneous, depending on the particular mix of Silicon Objects chosen for the array. These Silicon Objects include multipliers, and other functional units. The architecture also supports various I/O standards including high speed serial I/O as well as internal RAM. FPOA supports 21-bit busses to communicate 16 data bits, a one-bit data-valid flag, and 4 control/state bits. The functionality of the control bits depends on the Silicon Object itself.

As shown in Figure 3.6, each Silicon Object can communicate directly with its 8 immediate neighbors. And with one level of pipelining this can be extended to be 24 other cells. Using more levels of pipelining, a Silicon Object's output signals can reach the rest of the array.

The ARRIVE (ARm microprocessor with Reconfigurable Instruction-flow controlled Vliw Extension) architecture introduced in [Zabe06],is considered a DSP oriented CGRA architecture. It introduced a high level of Instruction Level Parallelism (ILP), flexibility and scalability. The structure is based on an enhanced RISK processor (ARM7 processor). This processor is tightly coupled with an reconfigurable ALU array, a vector load/store unit and a control flow manipulation unit. This structure also supports context switching between different configurations. The structure is introduced as a VHDL RTL model. The authors also introduce an architecture exploration framework for their CGRA. The structure of the arrive CGRA is shown in Figure 3.7. The ARM processor is connected to the ALU array through the processor bus. The ALU array is connected in a

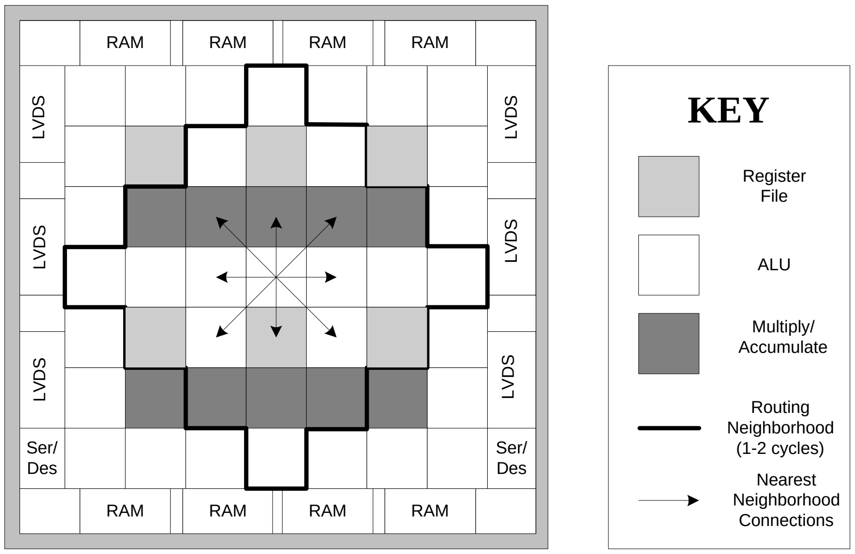

RAM
LVDS
Ser/ Des
KEY
Register File
ALU
Multiply/ Accumulate
Routing Neighborhood (1-2 cycles)
Nearest Neighborhood Connections

Figure 3.6: MathStar's FPOA Architecture

two dimension configuration.

Table 3.2 gives a comparison between different architectures presented in this section. CGRAs provide the configurability of FPGA with a lower configuration effort (data required for configuration are smaller in size, and short configuration time), and higher performance close to that of ASIC. CGRA that support run-time reconfiguration will be a appropriate choice for implementing DSP systems. CGRAs support DSP application through providing programmable ALUs in their PEs. The support of configurable blocks that perform DSP operations (MAC, FFT) requires investigation. The support of parallelism, which is required by almost every DSP application, is through the availability of different resources that can operate in parallel (LUTs , ALUs, Processors). The support of parallel processing through instruction set parallelism and multi-threading is also of interest.

Name	Source	Array	PE	Support Runtime Reconfiguration	Processor	Support DSP
Raw	[Tayl02, Wain97]	2D Array	MIPS CPU + Cach + 32 FPU	No	32-bits MIPS-like	No
PipeRench	[Gold00, Schm02]	1D Array of Strips	Shifter + Multiplixors + LUT + Register File	Yes	None	No
RaPiD	[Ebel96, Cron99, Cron98, Ebel97]	1D Common Bus	Coarse-grain blocks (ALU, FFT, DCT)	Yes	None	Yes
PACT XPP	[Baum03, GmbH]	Hierarchal 2D Structure	ALU + Reg file	Yes	None	No
MathStar	[Helg03]	2D Array	ALU or Multiplier	No	None	No
ARRIVE	[Zabe06]	ARM + 2D Array	ALU + Reg	No	ARM	Yes

Table 3.2: CGRA Summary

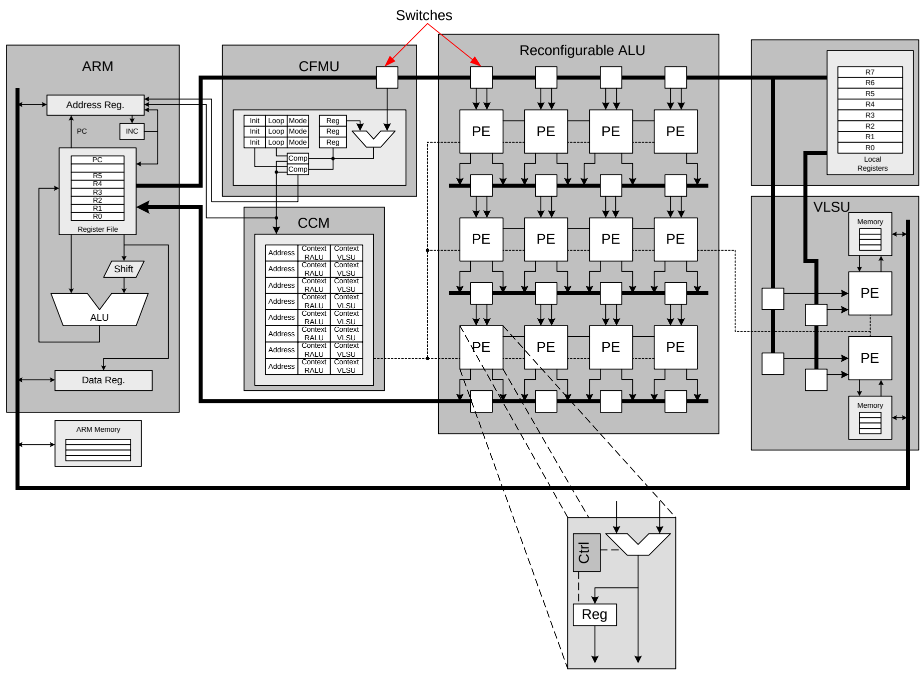


Figure 3.7: ARRIVE Architecture

3.7 Summary

In this chapter, a literature review of the architecture exploration process was presented. According to the Y-chart scheme introduced in section 2.3.1 different issues in the domain of architecture exploration were highlighted. First, different methodologies for architecture exploration were reviewed. Next, miscellaneous approaches used in architecture exploration frameworks to search the design space were introduced and several support tools for architecture exploration were given. Methods to evaluate embedded systems architectures were also explored. Frameworks for architecture exploration of embedded system is then reviewed. Finally, a review of several CGRA as possible architectures platforms was given. From the literature review the following can be concluded:

- The concept of orthogonalization of concerns [Keut00] is crucial for architecture exploration. It enables separating the exploration process into phases depending on different concerns. This concept was the base of many methodologies for architecture exploration.

- A review of different techniques to search the design space are given:

    1. Exhaustive search is used in architecture exploration to find the optimal architecture. Due to its time consuming nature, iterative search is modified to give suboptimal solutions in a reasonable time.

    2. Local search is a simple heuristic approach used in architecture exploration. It gives results in a reasonable time, however solutions obtained are suboptimal. This approach is used in architecture exploration combined with a priori analysis of the application to limit search in the design space.

   3. Meta-heuristic approaches is the form of evolutionary algorithms are commonly used as a search technique in architecture exploration. Two commonly used algorithms are SPEA2 and NSGA-II, the former has a simpler computational model. Evolutionary algorithms have some edge over local search techniques by covering a wide range of alternative solutions quickly.
   4. Other meta-heuristic algorithms such as tabu search and simulated annealing have a limited use in architecture exploration.

- Three different categories of evaluation techniques were reviewed, accurate simulation, statistical simulation and analytical evaluation. A combined statistical-analytical evaluation methodology seems to be a robust way to perform efficient evaluation in a reasonable time.

- The review of several CGRA architectures indicate that they share the following common useful features:

   1. Provide coarse-grained blocks for efficient implementations of different applications. These blocks range from simple LUTs to a complete processor.
   2. Support runtime reconfiguration which enables implementing large application on a single chip. This support is through different reconfiguration management techniques.
   3. The support of DSP is limited. CGRAs support DSP application through providing programmable ALUs in their PEs. The support of configurable blocks that perform DSP operations (MAC, FFT) requires investigation.
   4. The support of parallelism, which is required by almost every DSP applica-

tion, is through the availability of different resources that can operate in parallel (LUTs , ALUs, Processors). The support of parallel processing through instruction set parallelism and multi-threading is limited.

The design of a DSP oriented CGRA that can support runtime reconfiguration should provide an efficient platform for architecture exploration for such applications.

3.7.1 Research Directions

Conclusions obtained from previous subsection can be used to guid our research in the following directions:

1. **The Design of an Architecture Exploration Framework for DSP application**
   In this part of the project the following research points will be investigated:
   (a) Investigate the exploration of the design space of DSP applications and their mapping to a heterogenous embedded architectures. Evolutionary algorithms will be investigated to be the tool to search the design space. Evolutionary algorithms could be combined with other algorithms to improve the performance.
   (b) Exploring the different architectures will include exploring both the computational sub-system and the communication sub-system (buses and direct links). The exploration of the computation and communication sub-system could be combined or performed separately. Separating the exploration of the two sub-system has the advantage of reducing the size of the design space to be explored. Both options will be investigated.

   (c) During the exploration process each point in the design space represents a solution architecture. An evaluation strategy that gives fast and accurate results should be used. A hybrid analytical-statistical approach will therefore be investigated. Using intelligent methods such as neural networks to improve the accuracy of analytical evaluation will be studied.

2. **The design of DSP oriented CGRA**

   In this part the following will be investigate:

   (a) Different implementation technologies could be the target of the architecture exploration process. The support of DSP applications in modern CGRA is limited. In this research project the implementation of DSP oriented CGRA will be investigated. The support for DSP will be through including coarse grained blocks of commonly used operators in DSP applications.

   (b) DSP applications are parallel in nature and requires processors that support parallelism at different levels (instruction level, multi-threading and processor level). In this project the implementation of CGRA that support these features will be investigated.

   (c) Integrating the architecture exploration framework of part two with the proposed CGRA architecture will be studied to map DSP application.

Chapter 4

Current Proposed Approaches

In chapter 3 a review of several architecture exploration frameworks, search techniques and evaluation techniques is given. The review shows that multi-objective evolutionary algorithms are efficient tools to search the design space for near optimal results. It also shows that statistical and analytical based evaluation is suitable for architecture exploration at high level of abstraction as they provide quick results with a reasonable accuracy.

In this chapter the preliminary results for an architecture exploration framework based on SPEA2 is introduced. The input to the design exploration too is a graph that represents the application. The output of the framework is a possible architecture that is considered to be a near optimal hardware implementation of the given application. The resulting architectures are composed of components chosen from an experimental core library estimated from real implementations. The core library used in the preliminary results contains three processors and a general representation of special function cores. It also contains different communication channels and buses. The details of the

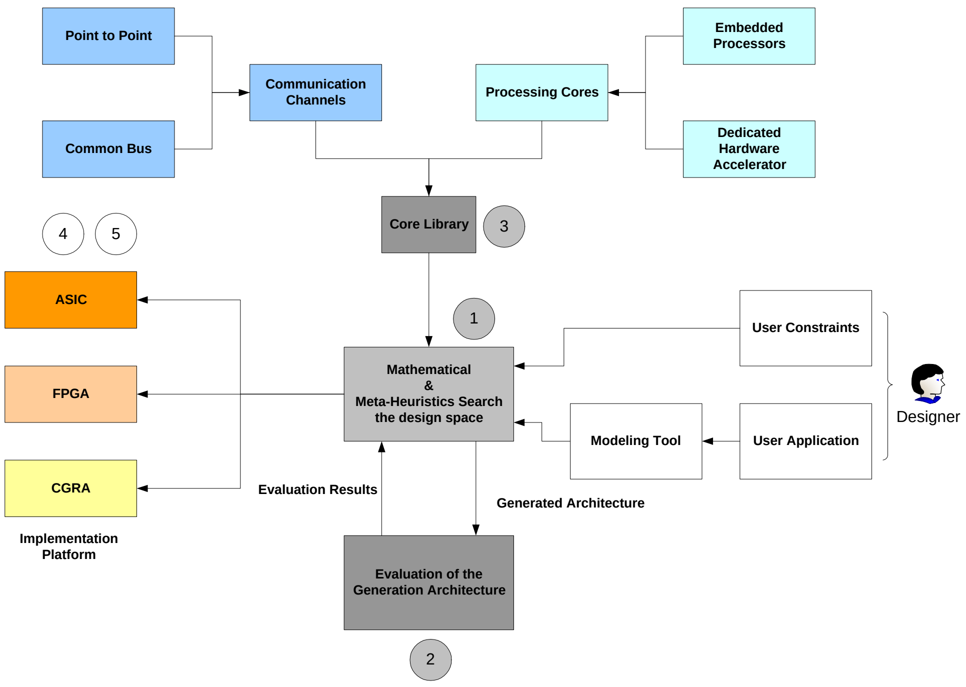


Figure 4.1: AE framework

implementation and the results are described in the following subsections.

Figure 4.1 shows the general architecture exploration framework introduced in chapter 1 with shaded circles indicating what has been implemented in this chapter.

4.1 Core Library

The core library contents are shown in Table 4.1 and Table 4.2. Table 4.1 contains information about processing cores included in the library. This library represents the platform used for computation and communication sub-systems. For each core four metrics are provided for four different attributes. These metrics include relative speed, area, power consumption and flexibility. Flexibility represents the ability of modifying the imple-

Core	Speed	Area	Power	Flexibility
PPC	1.2	0.6	0.5	0.95
RCore	1.9	0.7	0.1	0.8
MB	1.4	0.6	0.8	0.95
HW-Core	0.1	0.5	0.2	0.2

Table 4.1: Core Library Contents - Processing Cores

Core	Speed	Area	Power
OBP	0.4	0.1	0.6
FSL	0.1	0.4	0.1
LMB	0.1	0.3	0.5
SPB	0.2	0.2	0.2

Table 4.2: Core Library Contents- Communication Channels

mentation. All these metrics are assumed to be in per-instruction units. The “HW-Core” represents a general model for custom IP cores.

Table 4.2 includes information about communication channels that can be used to connect different components of the resulting systems. Three measures are available for each communication channel. These measures are also relative and per instruction unit. “SPB” stands for Special Peripheral Bus, which is a general model for custom bus depending on the implemented hardware.

4.2 Implementation Using ECJ

ECJ [Labo06] is a Java-based evolutionary computation framework. It enables building different applications that uses Genetic Algorithms (GA) in their operation. The framework provides a set of Java classes that are used to define each GA operation (Selection, cross-over, mutation, evaluation). The problem is defined through a parameter file that contain a set of parameters for each operation of the evolutionary algorithms. Each class of the framework reads a set of parameters that control its behavior. The problem is defined by writing one or more classes and overriding the behavior of the base class for a specific problem. At least a class inherited from a class called Problem should be written to specify the evaluation scheme for each individual in the generation [Labo06]. By writing new classes , new set of parameters can be added to define the specific problem. The framework contains a built-in parser that is used to parse the parameter file for a specific problem. Each class in the framework starts by a setup phase the makes use of this parser to extract its specific parameters. The framework contains a GUI for loading the parameter file and displaying the results. ECJ contains an implementation for SPEA2. This implementation is used to perform architecture exploration within the framework.

In this work, three classes are added to define the architecture exploration problem. First a class based on the IntegerVectorIndividual class. It is called ExplorationIndividual. This class is used to represent an individual in a given generation as a vector of integer numbers. The class is modified from its parent to interpret the genomes in the individual chromosome into the architecture it represents and displays it to the user.

The second class is based on the SPEA2MultiObjectiveFitness. It is called ExplorationFitness. This class is used to define the stoping criteria for the evolutionary process. It represents the SPEA2 fitness beside four problem specific finesses , one for each objective, to be optimized. The four objectives are performance, area, power consumption, and flexibility.

The third and the most important class is based on the Problem class. It is called ExplorationProblem. This class is used to evaluate each individual. The class has a member function called evaluate which is called for each individual to be evaluated. In this problem each individual represents a specific architecture that is a solution for the

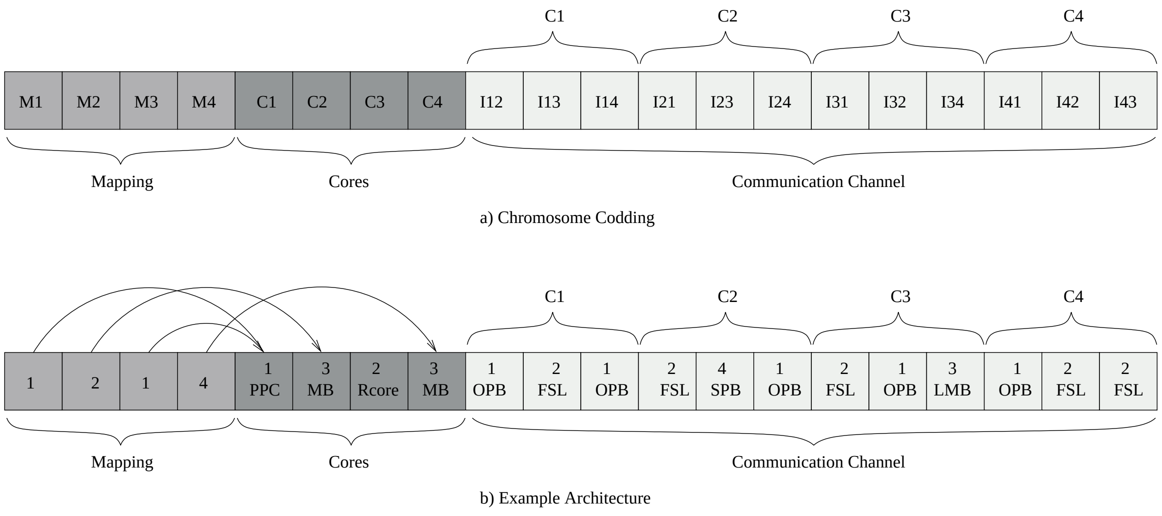


Figure 4.2: Chromosome Representation

given problem. The problem is defined in the parameter file. The problem is described as the number of software blocks and properties for each block results from profiling the given application. It is assumed that these properties are the size (number of instructions) and the number of iterations. The evaluate use the problem definition described above with a hard-coded definition for the core library defined in section 4.1.

4.3 Chromosome Representation

The chromosome coding used in this work is shown in Figure 4.2 for four software blocks problem. As shown first four genes are used for mapping of the software blocks on the architecture cores. The next four genes are used to define the cores (Note: core 3 is not used) and the remaining genes are used to define the communication channels between each core and the other (Note: communication channels for unused cores are ignored). Changing the chromosome's genes values give a new architecture and software mapping into it. The chromosome length (cl) is calculated from the number of software blocks sb using Equation (4.1).

$$cl = (sb \times 2) + (sb - 1) \times sb \tag{4.1}$$

4.4 Analytical Evaluation Scheme

Analytical approach is used for architecture evaluation. The architecture represented in the chromosome is evaluated using the metrics of the cores as found in the library introduced in Section 4.1. Each chromosome gene is decoded and the information about the architecture are extracted. For each software block, the corresponding IP core it is mapped to is selected, from the core library and the information about that core is used to calculate four cost function for each objective. If the core is not used for any software block it does not affect the cost function. The performance cost calculation depends on how the software blocks are mapped to the IP cores. Area and Flexibility cost functions are the sum of those of each used IP core. The power cost function is the sum of the effect of each software core. The same calculation is performed for the communication cost. The total cost functions is the sum of all the cost functions. The fitness for each objective is then calculated from the cost function.

Equation 4.2 is a formulation of the software model used in the proposed framework and represented using graphes as shown in Figure 4.3. Each software block $S(k)$ has size s_k. For each pair of software blocks $S(k_1)$ and $S(k_2)$, software block $S(k_1)$ makes i_m calls to $S(k_2)$.

$$\begin{aligned} S(k) &= \{s_k\} \\ i_m &= \{S(k_1) \longrightarrow S(k_2)\} \end{aligned} \tag{4.2}$$

Equations 4.3 and 4.4 is a formulation of the experimental library introduced in section 4.1. Each core $C(i)$ is represented by four parameters tc_i for speed, ac_i for area, fc_i for flexibility and pc_i for power consumption. The bus $B(j)$ is represented by three parameters tb_j for speed, ab_j for area and fb_j for flexibility.

$$C(i) = \{tc_i, ac_i, fc_i, pc_i\} \tag{4.3}$$

$$B(j) = \{tb_j, ab_j, fb_j\} \tag{4.4}$$

The mapping of the software application into hardware, is expressed by equation 4.5. Each software block $S(k)$ is mapped to a hardware block $C(i)$. If the software block mapped to $C(i_1)$ calls another software block mapped to $C(i_2)$ then the bus $B(j)$ is used for communication between the two hardware cores.

$$\begin{aligned} S(k) &\longrightarrow C(i) \\ B(j) &= \{C(i_1) \longrightarrow C(i_2)\} \end{aligned} \tag{4.5}$$

Equation 4.6 is used to calculate the overall performance values for the used hardware cores, while equation 4.7 is used to calculate the performance values for communication busses.

$$\begin{aligned}
T_c &= max\left(\sum_k tc_i \times s_k\right) \\
A_c &= \sum_i \sum_k ac_i \times s_k \\
F_c &= \sum_i \sum_k fc_i \times s_k \\
P_c &= \sum_i \sum_k pc_i \times s_k
\end{aligned} \tag{4.6}$$

$$\begin{aligned}
T_b &= \sum_j \sum_k tb_j \times i_m \\
A_b &= \sum_j \sum_k ab_j \times i_m \\
P_b &= \sum_j \sum_k pb_j \times i_m
\end{aligned} \tag{4.7}$$

Equation 4.8 is then used to calculate the four cost functions used for the evaluation of the generated architecture.

$$\begin{aligned}
max(f_1 &= T_c + T_b) \\
max(f_2 &= A_c + A_b) \\
max(f_3 &= F_c) \\
max(f_4 &= P_c + P_b)
\end{aligned} \tag{4.8}$$

4.5 Preliminary Results

Preliminary results were obtained by the proposed framework using the test-bench are shown in Figure 4.3. The test-bench consists of 6 software blocks. Specification-Level Intermediate Formate (SLIF) [Vahi95] graph is used to represent the given application. The design is entered to the framework using a file which includes a description for each node and edge. It gives the user the flexibility to set constraints for the implementation (such as timing requirements). Each node represents a software block, and the edge represents dependency relationship. An edge between software block A and software block B means that software block A is calling software block B, and the number represent the number of times the call is made which are extracted from profiling the given application. Each software block has some attributes. In the simple model used in the primarily test these attributes are the size of the software block.

The system is evaluated with test-bench of figure 4.3. Different runs are made, each with 400 generations, each contains 200 individuals. In each run the framework investigates 80000 possible architectures. The best individual of the run according to SPEA2 fitness is then selected as output. The average fitness for each objective and for SPEA2 fitness across each generation are plotted for each run.

Figure 4.4 shows the result of a run with full optimization for the four objectives and with no constraints. As shown area, performance and power consumption improve on average with each generation, while the flexibility is reduced. Increasing the over-all performance of the system reduces its flexibility.

As shown in Figure 4.5 the tool selected two PPCs to implement the architecture.

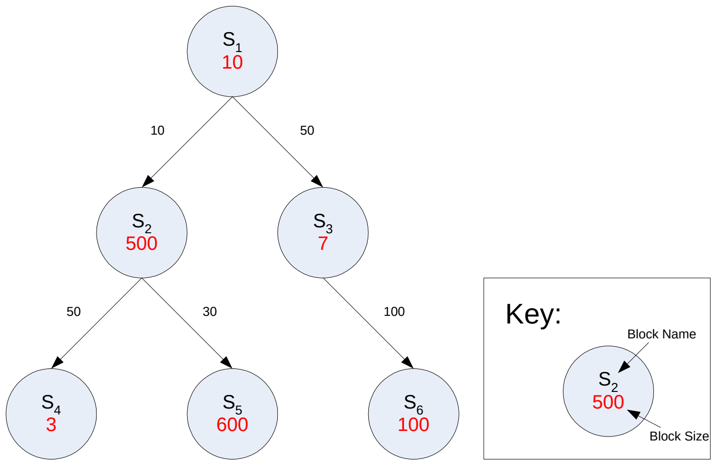


Figure 4.3: Test-bench used for the preliminary results

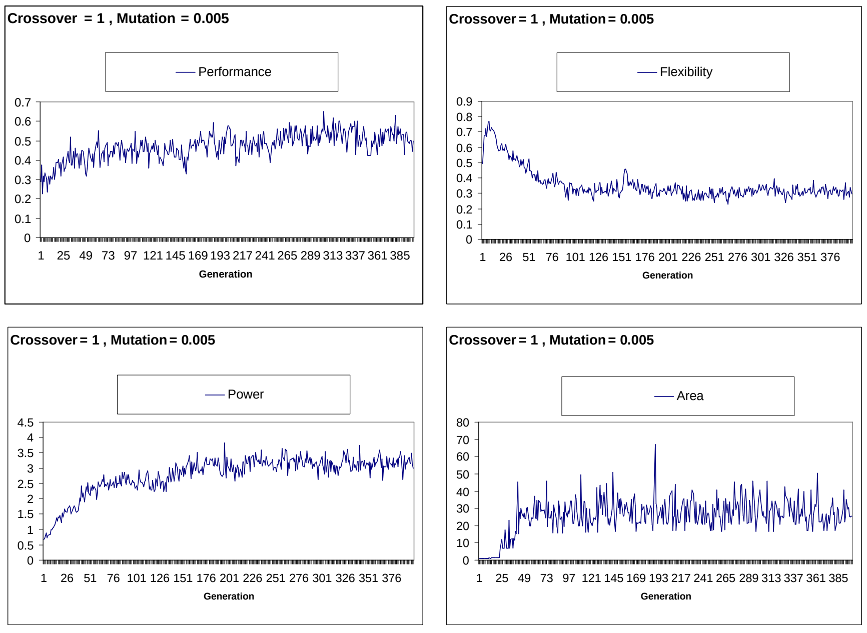
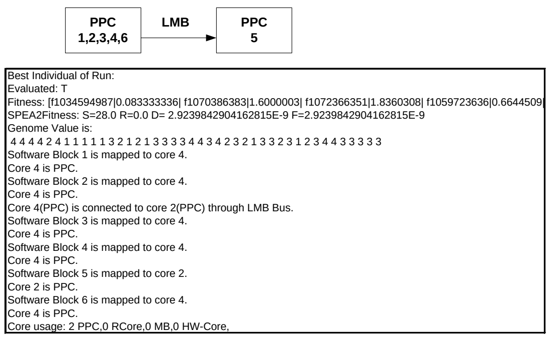


Figure 4.4: Results of AE without any constraint, full optimization



```
Best Individual of Run:
Evaluated: T
Fitness: [f1034594987|0.083333336| f1070386383|1.6000003| f1072366351|1.8360308| f1059723636|0.6644509|
SPEA2Fitness: S=28.0 R=0.0 D= 2.9239842904162815E-9 F=2.9239842904162815E-9
Genome Value is:
 4 4 4 4 2 4 1 1 1 1 1 3 2 1 2 1 3 3 3 3 4 4 3 4 2 3 2 1 3 3 2 3 1 2 3 4 4 3 3 3 3 3
Software Block 1 is mapped to core 4.
Core 4 is PPC.
Software Block 2 is mapped to core 4.
Core 4 is PPC.
Core 4(PPC) is connected to core 2(PPC) through LMB Bus.
Software Block 3 is mapped to core 4.
Core 4 is PPC.
Software Block 4 is mapped to core 4.
Core 4 is PPC.
Software Block 5 is mapped to core 2.
Core 2 is PPC.
Software Block 6 is mapped to core 4.
Core 4 is PPC.
Core usage: 2 PPC,0 RCore,0 MB,0 HW-Core,
```

Figure 4.5: Resulting Architecture, for full optimization

Five out of the six blocks given in the test-bench are mapped to the first PPC, while the final block (has the larger size) is running separately on the second PPC. LMB is used for communication between the two processors. The architecture proposed by the framework is at a high level of abstraction. More optimization can be made in lower levels through out the implementation process. Note that the performance and flexibility are not well optimized compared to the other two objectives, since the optimization of performance contradict with optimization of the flexibility given the library introduced earlier.

The framework gives the user the ability to turn off/on the optimization for any of the four objectives. Figure 4.6 shows the results of the run with performance optimization turned off. As shown the over-all SPEA2 fitness is improved. This run results the same architecture shown in Figure 4.5.

Figure 4.7 shows the results after turning the power optimization off. Note when the power optimization was turned off, the performance optimization improved relatively. The resulting architecture is the same that of the previous two runs (shown in Figure 4.5).

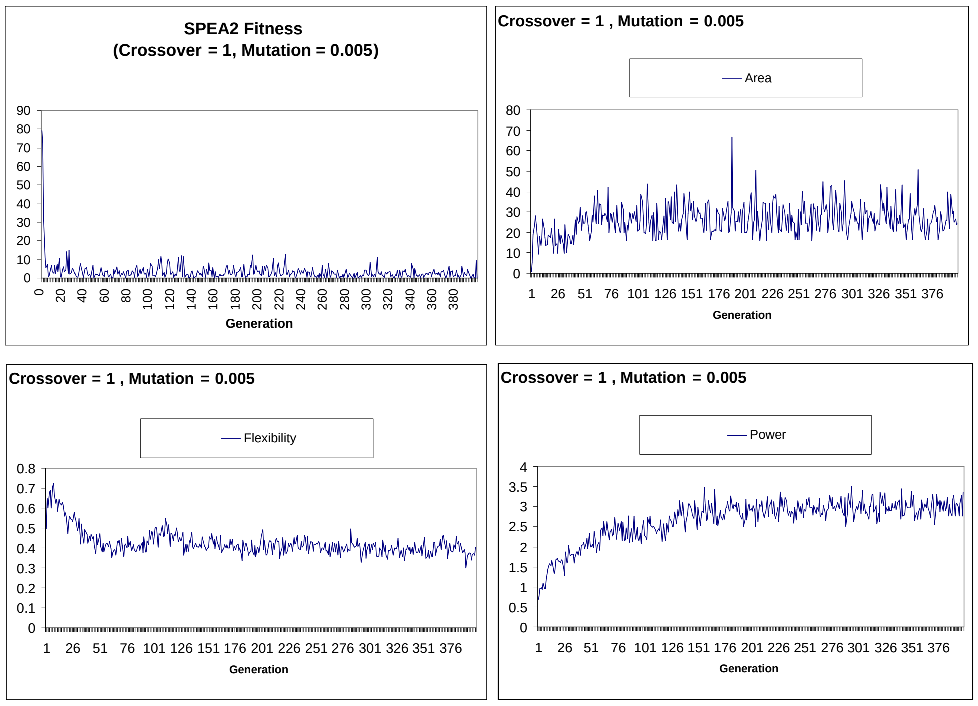


Figure 4.6: Results of AE without performance optimization

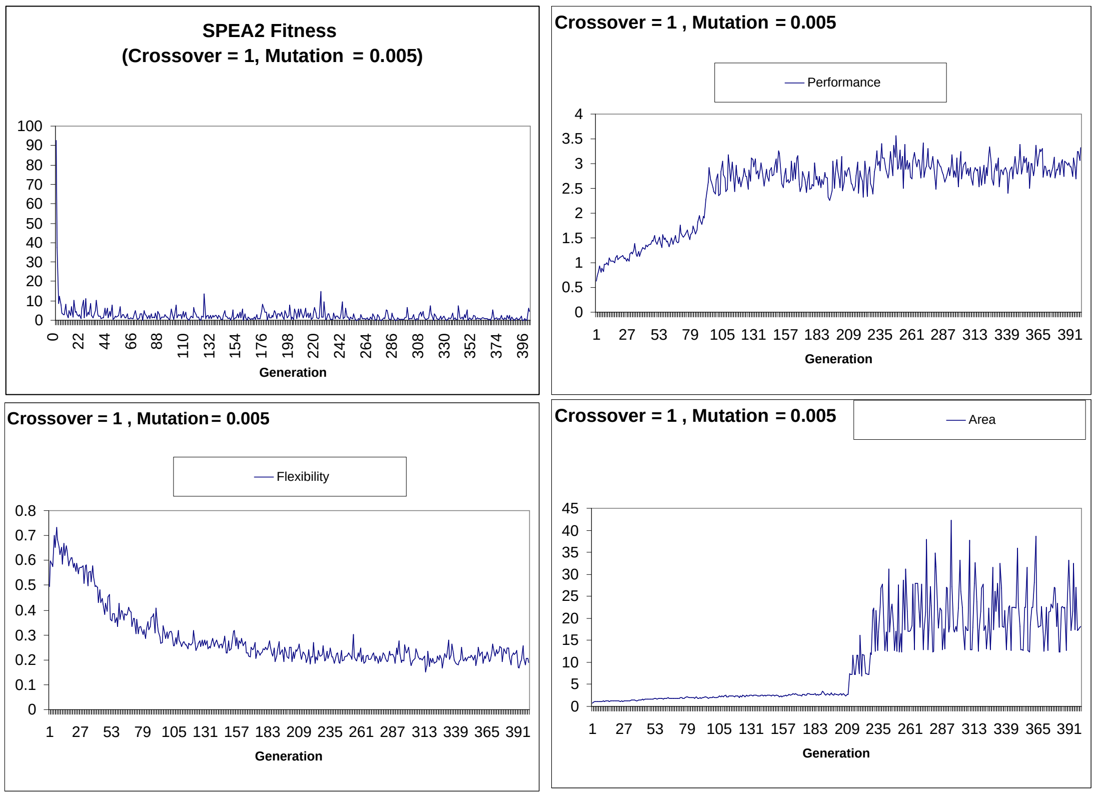


Figure 4.7: Results of AE without power optimization

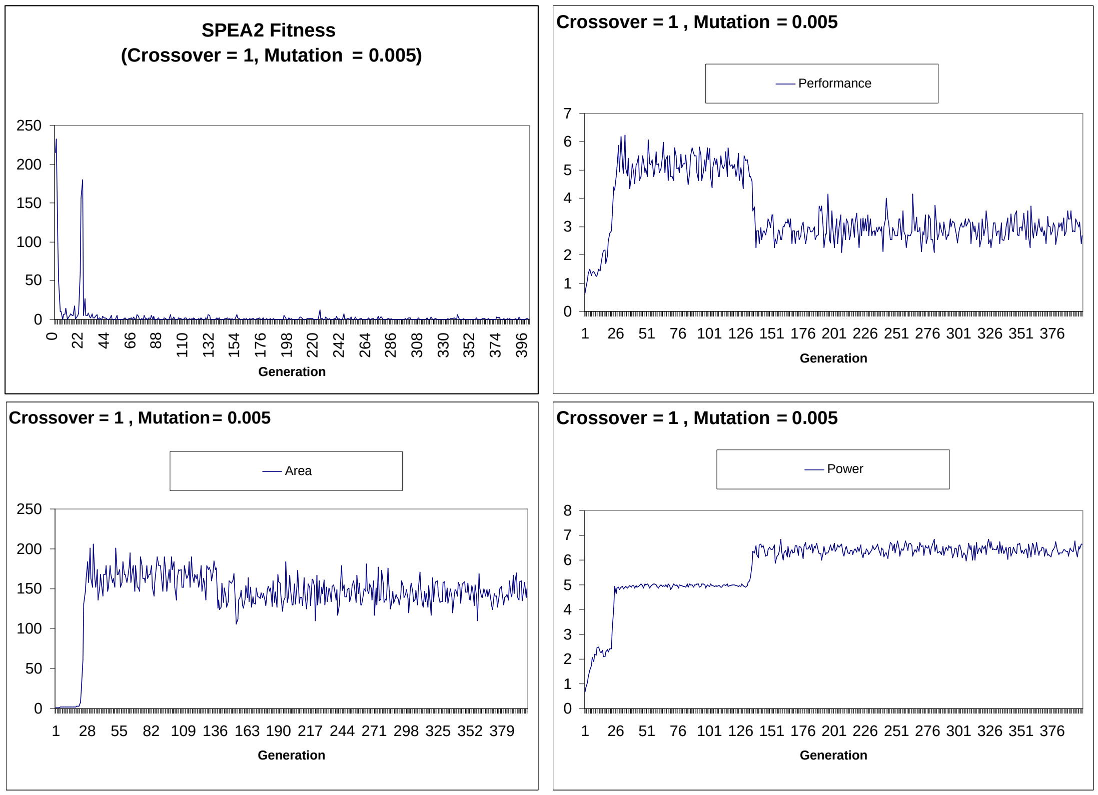
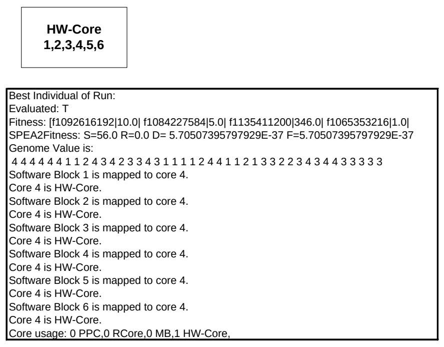


Figure 4.8: Results of AE without flexibility optimization

Turning the flexibility optimization off results in a good optimization of all the other three objectives as shown in Figure 4.8. The resulting architecture is shown in Figure 4.9. It is clear that when the flexibility is turned off the system is all implemented in pure HW. That is because HW-core gives the best metrics in the library except for flexility.

Turning the area optimization off result in a much larger architecture consisting of 3 processors as shown in Figure 4.11. The larger blocks are given a separate processor (PPC). The other three blocks are mapped to a single processor (RCore). Communication between the three processor is through FSL links. Note that the communication channels are created to communicate only in the direction of the communication given in the test-



```
Best Individual of Run:
Evaluated: T
Fitness: [f1092616192|10.0| f1084227584|5.0| f1135411200|346.0| f1065353216|1.0|
SPEA2Fitness: S=56.0 R=0.0 D= 5.70507395797929E-37 F=5.70507395797929E-37
Genome Value is:
 4 4 4 4 4 4 1 1 2 4 3 4 2 3 3 4 3 1 1 1 1 2 4 4 1 1 2 1 3 3 2 2 3 4 3 4 4 3 3 3 3 3
Software Block 1 is mapped to core 4.
Core 4 is HW-Core.
Software Block 2 is mapped to core 4.
Core 4 is HW-Core.
Software Block 3 is mapped to core 4.
Core 4 is HW-Core.
Software Block 4 is mapped to core 4.
Core 4 is HW-Core.
Software Block 5 is mapped to core 4.
Core 4 is HW-Core.
Software Block 6 is mapped to core 4.
Core 4 is HW-Core.
Core usage: 0 PPC,0 RCore,0 MB,1 HW-Core,
```

Figure 4.9: Resulting Architecture, for flexibility optimization off

bench. The average fitness of the run are shown in Figure 4.10. Note that the flexibility fitness is improved because of the use of more processors.

The framework enables adding constraints to a specific block of the input application. The test-bench is modified as shown in Figure 4.12. Timing constraints are added to blocks 1,2 and the results of this run are shown in Figure 4.13. Note that the performance is not well optimized because of the timing constraints added to the test-bench.

The resulting architecture is shown in Figure 4.14. As shown 2 processors and one HW accelerator is used. The two constrained blocks are implemented in two separate modules to operate in parallel to meet the timing constraints. For the two larger blocks, one is implemented in a separate units (block 4 is mapped to a separate PPC), and the other one is mapped to the hardware accelerator.

The results obtained show that the proposed architecture exploration framework efficiently explore the design space for a heterogenous architectures. The framework investigate thousands of architectures from the design space. The framework give different solutions for a given problem according to the optimization method and the user constraints.

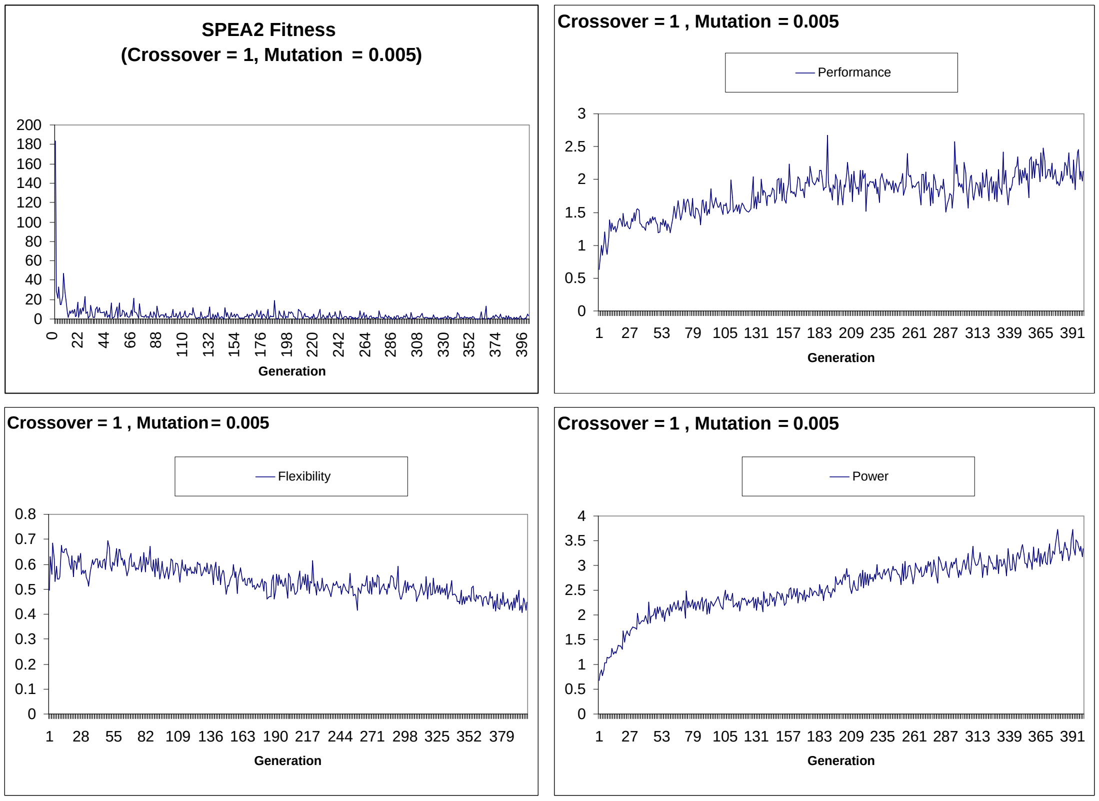
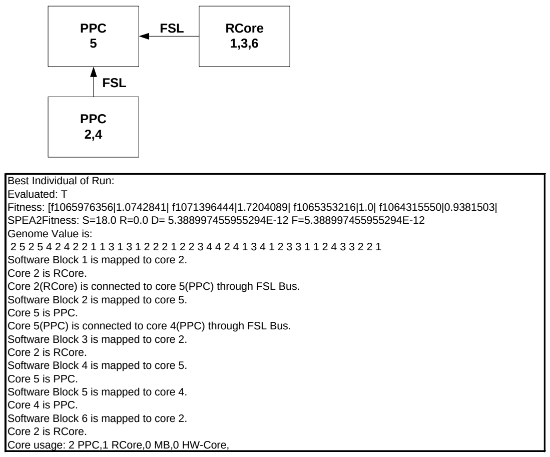


Figure 4.10: Results of AE without area optimization



```
Best Individual of Run:
Evaluated: T
Fitness: [f1065976356|1.0742841| f1071396444|1.7204089| f1065353216|1.0| f1064315550|0.9381503|
SPEA2Fitness: S=18.0 R=0.0 D= 5.388997455955294E-12 F=5.388997455955294E-12
Genome Value is:
 2 5 2 5 4 2 4 2 2 1 1 3 1 3 1 2 2 2 1 2 2 3 4 4 2 4 1 3 4 1 2 3 3 1 1 2 4 3 3 2 2 1
Software Block 1 is mapped to core 2.
Core 2 is RCore.
Core 2(RCore) is connected to core 5(PPC) through FSL Bus.
Software Block 2 is mapped to core 5.
Core 5 is PPC.
Core 5(PPC) is connected to core 4(PPC) through FSL Bus.
Software Block 3 is mapped to core 2.
Core 2 is RCore.
Software Block 4 is mapped to core 5.
Core 5 is PPC.
Software Block 5 is mapped to core 4.
Core 4 is PPC.
Software Block 6 is mapped to core 2.
Core 2 is RCore.
Core usage: 2 PPC,1 RCore,0 MB,0 HW-Core,
```

Figure 4.11: Resulting Architecture without area optimization

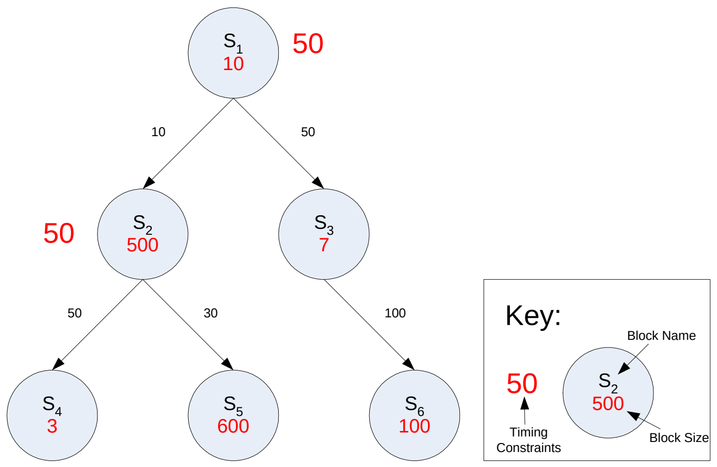


Figure 4.12: Constrained Test-bench used for the preliminary results

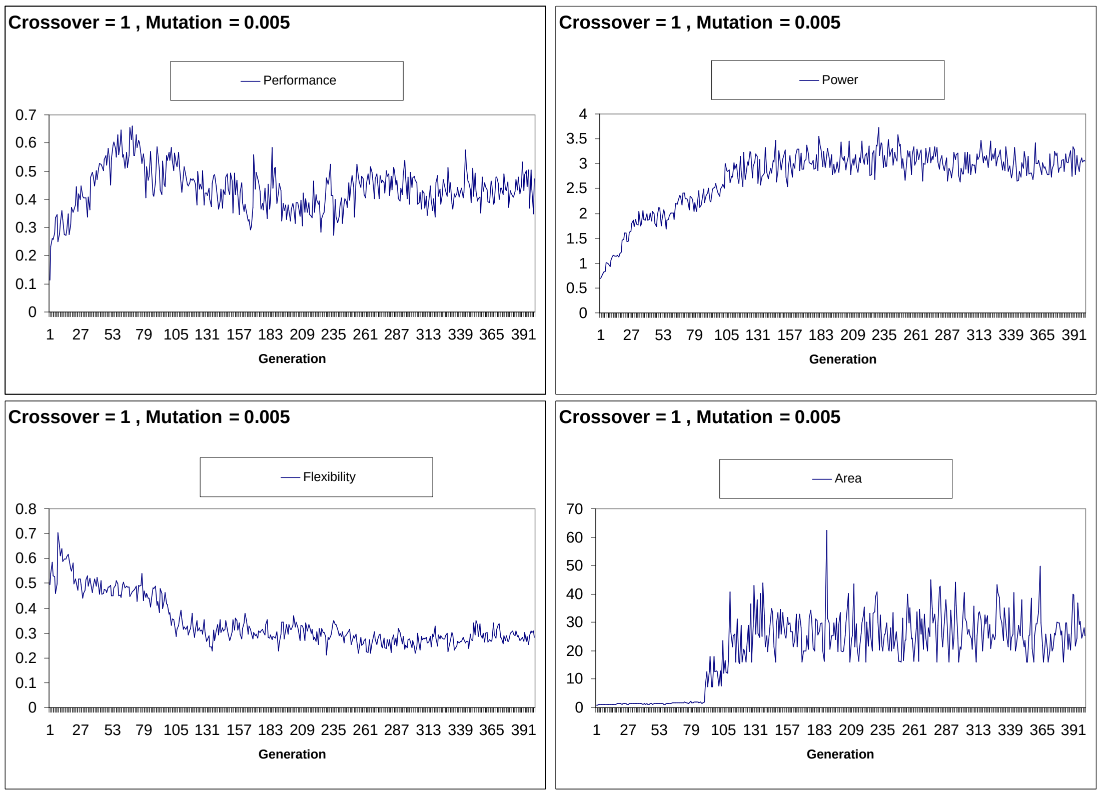
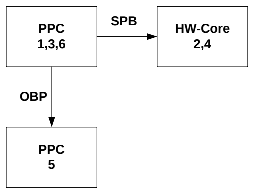
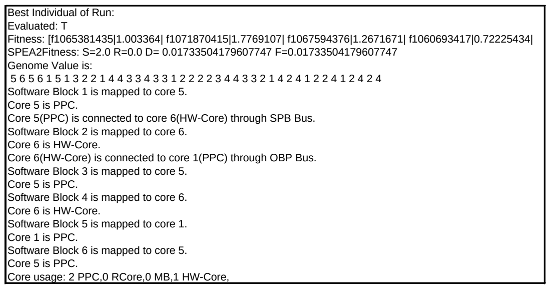


Figure 4.13: Results of AE with constraints





```
Best Individual of Run:
Evaluated: T
Fitness: [f1065381435|1.003364| f1071870415|1.7769107| f1067594376|1.2671671| f1060693417|0.72225434|
SPEA2Fitness: S=2.0 R=0.0 D= 0.01733504179607747 F=0.01733504179607747
Genome Value is:
 5 6 5 6 1 5 1 3 2 2 1 4 4 3 3 4 3 3 1 2 2 2 2 3 4 4 3 3 2 1 4 2 4 1 2 2 4 1 2 4 2 4
Software Block 1 is mapped to core 5.
Core 5 is PPC.
Core 5(PPC) is connected to core 6(HW-Core) through SPB Bus.
Software Block 2 is mapped to core 6.
Core 6 is HW-Core.
Core 6(HW-Core) is connected to core 1(PPC) through OBP Bus.
Software Block 3 is mapped to core 5.
Core 5 is PPC.
Software Block 4 is mapped to core 6.
Core 6 is HW-Core.
Software Block 5 is mapped to core 1.
Core 1 is PPC.
Software Block 6 is mapped to core 5.
Core 5 is PPC.
Core usage: 2 PPC,0 RCore,0 MB,1 HW-Core,
```

Figure 4.14: Resulting Architecture for the constrained Test-Bench

4.6 Summary

In this chapter, a framework for architecture exploration based on SPEA2 and analytical evaluation is introduced. The results of running a test-bench on the framework are given. The results show the ability of the framework to optimize multiple objectives. The framework enables the user to constraint some blocks of the input application. The proposed framework require modifications to add more details to the resulting architecture such as the memory structure and I/O communication. An experimental library is used by the framework to search the design space. Accurate library measures should be obtained to get more accurate results. Given these results, further investigations are required in the following points:

1. Investigate the effect of the crossover, selection and mutation operators on the exploration process.

2. Study the use of other MOEA such as NSGA-II for architecture exploration. Some

modifications may be required for the chromosome representation of the architecture for more efficient representation (The chromosome size increase exponentially with the application size).

3. Investigate combining MOEA with other search techniques to exploit different points in the design space may improve the fitness of the framework results.

4. Study combining different evaluation approaches with analytical evaluation to increase the accuracy of the evaluation phase.

Chapter 5

Proposed Approaches & Directions

In this chapter proposed approaches and research directions will be summarized. From the literature review presented in Chapter 3 the research in this thesis will tend to focus on two directions. The first is set to investigate the design of an architecture exploration framework for the design and implementation of embedded DSP systems. The framework's main goal is to generate near optimal architectures for a given DSP application. In Chapter 4 a framework based on SPEA-II evolutionary algorithms and a simple analytical evaluation scheme was implemented and some preliminary results were presented. Further enhancements and modifications are planned and will be further discussed in this chapter.

The second directions is geared towards the design of a Coarse Grained Reconfigurable Array (CGRA) chip optimized for DSP applications. The plan is to integrate it with the proposed framework for mapping DSP application efficiently. The support of parallelism at the architecture level will be also investigated.

Figure 5.1 shows the current research state and directions that will be discussed in

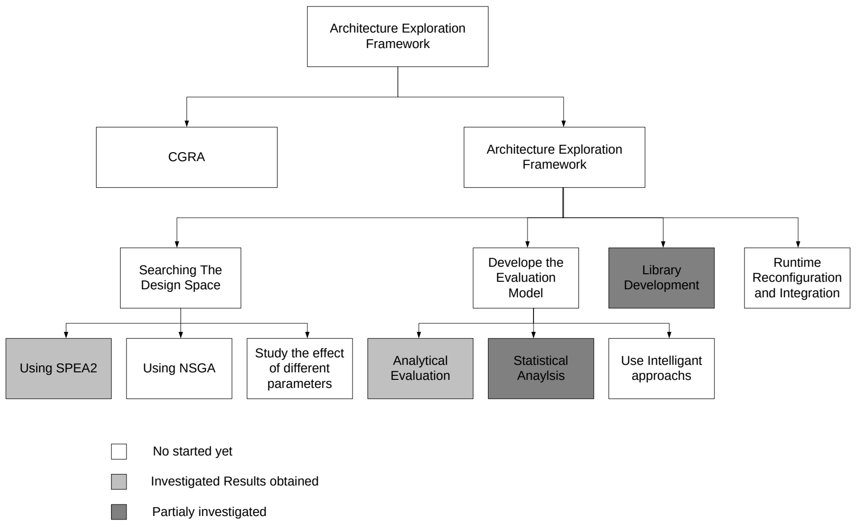

Architecture Exploration Framework
CGRA
Architecture Exploration Framework
Searching The Design Space
Develope the Evaluation Model
Library Development
Runtime Reconfiguration and Integration
Using SPEA2
Using NSGA
Study the effect of different parameters
Analytical Evaluation
Statistical Anaylsis
Use Intelligant approachs
No started yet
Investigated Results obtained
Partialy investigated

Figure 5.1: Research State and Directions

the following sections.

5.1 Architecture Exploration Framework

As presented in Chapter 4, the preliminary design exploration framework makes use of an experimental library to form the design space. SPEA-II multi-objective evolutionary algorithm is used for searching the design space, with two point crossover to generate new generations and different selection, and mutation rates. A preliminary evaluation techniques are used to evaluate different architectures during the search.

5.1.1 Searching the Design Space

The performance of the optimization tool SPEA-II can be further improved by experimenting with different crossover techniques and the population sizes used in each generation. A study of the effect of different parameter settings will be carried out. According to the literature review NSGA-II optimization tool produces better results at the cost of more computation time [Erba06]. The use of NSGA-II will also be studied and further modifications of both algorithms may be required. The chromosome representation of the architecture given in section 4.2 will need to be modified accordingly. More investigation for the crossover, selection and mutation effect on the exploration process will be performed.

The proposed framework performs exploration of both the computation and communication sub-systems concurrently. The design space formed by both computation and communication cores is complex and huge and searching it is time consuming. Separating the design space into two parts (computation and communication) and searching them in two different phases will be investigated.

5.1.2 Core Library

The library introduced in section 4.1 contains a set of GPP and IP cores that can be used to build a heterogenous embedded system and described at a very high level of abstraction. The metrics in the library should be accurate to reflect more realistic usable components. To gain this accurate metrics physical implementation should be made for DSP applications with different architectures. This allows compiling accurate information about the performance and power consumption of each component in a actual implementations. So far the library covers a small set of components. More components should be added to the library in order to give the tool more alternatives. Different IP cores from different vendors will be investigated and added to the library.

5.1.3 Evaluation Techniques

A simple analytical evaluation approach is used in the preliminary framework. Future work will be directed toward two main points:

1. Study the development of a hybrid analytical-statistical evaluation approach to be used during the exploration. This will be first investigated at a high level of abstraction to speed up the exploration process. Fine tuning of the resulting architecture could be performed at a lower level at a later stage.

2. Study the use of techniques such as fuzzy logic and neural networks to improve the accuracy of the evaluation phase [Hart01, Oyam04].

5.1.4 Integration with Implementation tools

The framework should be integrated with other developed tools as shown in Chapter 2 (Figure 2.6) to perform hardware implementation of the generated architecture. Integration implies generating the necessary files required by the implementation tools to perform physical realization of the design. The files include the source code for software block, HDL for IP cores and any other files required by the implementation tool. The result of this framework can be implemented either using FPGA or ASIC. The support for both flows will be investigated. Also in the second part of the project we propose the design of a DSP oriented CGRA. The support for this device will be also studied.

5.1.5 Runtime Reconfiguration

If the application to be implemented requires more resources than is available, the designer then will have to resort to one of two options: (i) add another chip which increases the cost of the design and affect the performance due to the communication between the two devices, (ii) employ runtime reconfiguration to dynamically replace unused portions of the device with other units. Adding runtime reconfiguration to the design requires adding a scheduling unit that is responsible for selecting the units to be replaced. The support for runtime reconfiguration in the framework will be investigated to be an alternative choice for implementing large designs.

5.2 Coarse Grained Reconfigurable Arrays (CGRA)

CGRAs provide a compromise solution between fine-grain reconfigurable devices such as FPGAs and ASIC if optimized for a specific application domain [Hart01]. The second research direction will be geared toward the design and implementation of a DSP based CGRA. A review of different reconfigurable architectures was introduced in chapter 3. The study indicates that CGRAs can support DSP application by providing programmable ALUs in their PEs. The support of configurable blocks that perform DSP operations such as MAC and FFT requires further investigation. Also the support of parallel processing is through the availability of different resources that can be configured to operate in parallel such as LUTs , ALUs, Processors. The support of parallel processing through instruction set parallelism and multi-threading is limited and requires further studying.

The following is a summary of what will be investigated:

1. Design of CGRA for DSP application. The CGRA will include explicit DSP blocks such as MAC and FFT. All of these blocks will be reconfigurable to operate in different modes with different data sizes. These blocks will be part of the reconfigurable array of Figure 5.2. The reconfigurable array could have the general structure shown in Figure 5.3.

2. Investigate adding support needed in the proposed CGRA for explicit parallelism at the architecture level as shown in Figure 5.2. This figure shows an example of two data paths attached with a configurable control unit. These two units could be configured through the reconfigurable switches and reconfigurable routing to operate as a single processor with two explicit threads, one VLIW machine, or two separate processors. Increasing the number of units will increase the level of parallelism that is supported.

3. High level synthesis is required to map DSP application to the proposed architecture. The framework proposed in the first phase could be modified to target the new CGRA architecture. As the components will be optimized for DSP application, accurate results can be obtained from the framework.

5.3 Work Plan

The work plan for this project is summarized in table 5.1. This give an approximate time line for the experimental work:

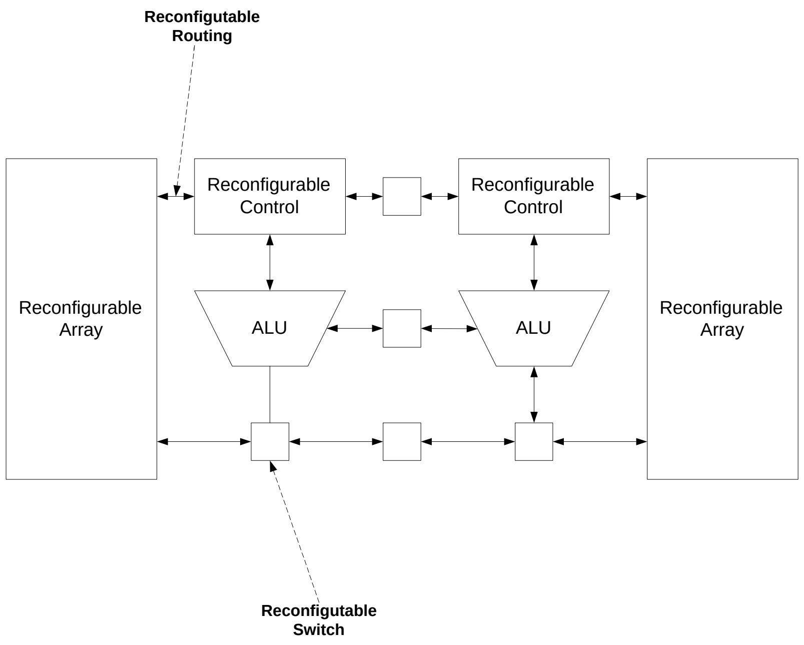


Figure 5.2: Proposed CGRA

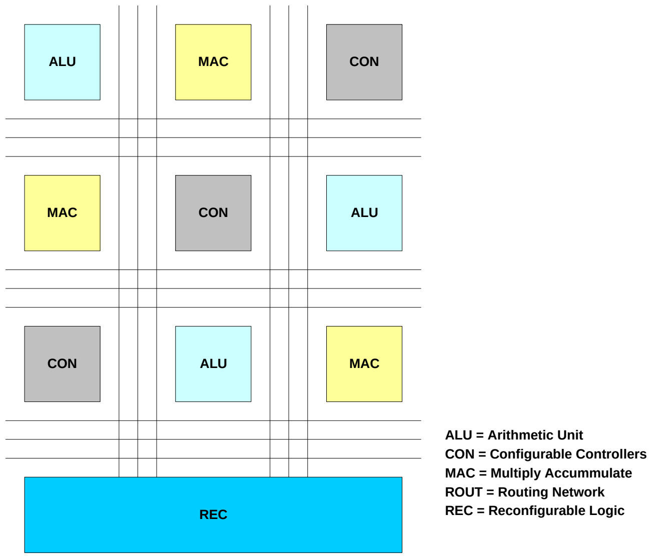

ALU
MAC
CON
MAC
CON
ALU
CON
ALU
MAC
REC
ALU = Arithmetic Unit
CON = Configurable Controllers
MAC = Multiply Accummulate
ROUT = Routing Network
REC = Reconfigurable Logic

Figure 5.3: Reconfigurable Array of Proposed CGRA

Number	Phase	Schedule	Description
1	Development of Architecture Exploration framework	W07 - F07	• Development of the optimization framework. • Development of the core library. • Development of the evaluation scheme. • Linking the framework to the physical implementation tools.
2	Runtime reconfiguration	F07	• Investigate using runtime reconfiguration to implement large DSP applications, and integrate that with the framework.
3	Implementation of a DSP oriented CGRA	F07-S08	• Design of the reconfigurable arrays. IP cores from different vendors could be used in this phase. • Design of the reconfigurable processing unit shown in Figure 5.2. • Physical implementation of the proposed architecture. • Modify the proposed architecture exploration framework to work with the new CGRA.

Table 5.1: Work Plan for the Project

Appendix A

Glossary

SoC	: System on Chip
DSE	: Design Space Exploration
MOEA	: Multi-Objective Evolutionary Algorithms
MOO	: Multi-Objective Optimization
VLIW	: Very Long Instruction Word
RISC	: Reduces Instruction Set Computer
FPGA	: Field Programmable Gate Array
CGRA	: Coarse-grained Reconfigurable Array
FPU	: Floating Point Unit
PE	: Processing Element

Bibliography

[Also00] A. Alsolaim, J. Becker, M. Glesner, and J. Starzyk, “Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems,” In *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines, 17-19 April 2000*, pp. 205–14, Dept. of Electr. & Comput. Eng., Ohio Univ., Athens, OH, USA, IEEE Comput. Soc, Napa Valley, CA, USA, / 2000.

[Asci01] G. Ascia, V. Catania, and M. Palesi, “Parameterised system design based on genetic algorithms,” In *Proceedings of IEEE 9th International Workshop on Hardware Software C-Design/CASHE, 25-27 April 2001*, pp. 177–82, Dipt. di Ingegneria Inf. e delle Telecommun., Catania Univ., Italy, ACM, Copenhagen, Denmark, / 2001.

[Asci04a] G. Ascia, V. Catania, and M. Palesi, “A ga-based design space exploration framework for parameterized system-on-a-chip platforms,” *Ieee Transactions on Evolutionary Computation*, vol. 8, No. 4, pp. 329–346, AUG 2004.

[Asci04b] Giuseppe Ascia, Vincenzo Catania, Maurizio Palesi, and Davide Patti, “Multi-objective optimization of a parameterized vliw architecture,” In *Proceedings - 2004 NASA/DoD Conference on Evolvable Hardware, Jun 24-26 2004*, pp. 191–198, IEEE Computer Society, Los Alamitos;Massey University, Palmerston, CA 90720-1314, United States;New Zealand, Seattle, WA, United States, 2004 Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[Asci05a] G. Ascia, V. Catania, and M. Palesi, “A multiobjective genetic approach for system-level exploration in parameterized systems-on-a-chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, No. 4, pp. 635–45, 04/ 2005.

[Asci05b] G. Ascia, V. Catania, M. Palesi, and D. Patti, “Exploring design space of vliw architectures,” In *16th International Conference on Application-Specific Systems, Architecture and Processors, 23-25 July 2005*, pp. 86–91, Dipt. di Ingegneria Informatica e delle Telecomunicazioni, Universita di Catania, Italy, IEEE, Samos, Greece, / 2005.

[Baks01] A. Bakshi, V. K. Prasanna, and A. Ledeczi, "Milan: a model based integrated simulation framework for design of embedded systems," In *Workshop on Languages, Compilers and Tools for Embedded Systems. (LCTES 2001), 22-23 June 2001*, pp. 82–7, Dept. of Electr. Eng. Syst., Univ. of Southern California, Los Angeles, CA, USA, ACM, Snowbird, UT, USA, 08/ 2001.

[Batt94] R. Battiti and G. Tecchiolli, "The reactive tabu search," *ORSA Journal on Computing*, vol. 6, No. 2, pp. 126–40, 1994.

[Baum03] V. Baumgarte, G. Ehlers, F. May, A. Nuckel, M. Vorbach, and M. Weinhardt, "Pact xpp - a self- reconfigurable data processing architecture," *Journal of Supercomputing*, vol. 26, No. 2, pp. 167–84, 2003.

[Bech03] A. Bechini, P. Foglia, and C. A. Prete, "Fine-grain design space exploration for a cartographic soc multiprocessor," *Computer Architecture News*, vol. 31, No. 1, pp. 85–92, 03/ 2003.

[Bell04] Robert H. Bell, Jr. Lieven Eeckhout, Lizy K. John, and Koen De Bosschere, ""deconstructing and improving statistical simulation in hls"," 2004.

[Bitt97] Ray Bittner and Peter Athanas, "Wormhole run-time reconfiguration," In *Proceedings of the 1997 ACM 5th International Symposium on Field-Programmable Gate Arrays, FPGA, Feb 9-11 1997*, pp. 79–85, Virginia Polytechnic Inst and State Univ, Blacksburg, VA, USA, ACM, New York, NY, USA, Monterey, CA, USA, 1997 Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[Bleu03] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "Pisa - a platform and programming language independent interface for search algorithms," In *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003. Proceedings, 8-11 April 2003*, pp. 494–508, Comput. Eng. & Networks Lab., ETH Zurich, Switzerland, Springer-Verlag, Faro, Portugal, / 2003.

[Broo00] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural -level power analysis and optimizations," In *Proceedings of 27th International Symposium on Computer Architecture, 10-14 June 2000*, pp. 83–94, Dept. of Electr. Eng., Princeton Univ., NJ, USA, ACM, Vancouver, BC, Canada, / 2000 Also available on CD-ROM in PDF format.

[Burg97] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *Computer Architecture News*, vol. 25, No. 3, pp. 13–25, 06/ 1997.

[Chak03a] S. Chakraborty, S. Kunzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," In *6th Design Automation and Test in Europe (DATE 03), 3-7 March 2003*, pp. 190–5, Swiss Fed. Inst. of Technol., Zurich, Switzerland, IEEE Comput. Soc, Munich, Germany, / 2003.

[Chak03b] S. Chakraborty, S. Kunzli, L. Thiele, A. Herkersdorf, and P. Sagmeister, "Performance evaluation of network processor architectures: combining simulation with analytical estimation," *Computer Networks*, vol. 41, No. 5, pp. 641–65, 04/05 2003.

[Chen92] D. C. Chen and J. M. Rabaey, "A reconfigurable multiprocessor ic for rapid prototyping of algorithmic-specific high-speed dsp data paths," *IEEE Journal of Solid-State Circuits*, vol. 27, No. 12, pp. 1895–904, 12/ 1992.

[Comp04] Katherine Compton and Scott Hauck, "Flexibility measurement of domain-specific reconfigurable hardware," In *ACM/SIGDA Twelfth ACM International Symposium on Field-Programmable Gate Arrays - FPGA 2004, Feb 22-24 2004*, pp. 155–161, Department of ECE, University of Wisconsin-Madison, Madison, WI 53706, United States, Association for Computing Machinery, Monterey, CA., United States, 2004 Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[Corp03] Intel Corporation, "Intel ixp 2400 network processor hardware reference manual. revision 7," November 2003.

[Cron98] D. C. Cronquist, P. Franklin, S. G. Berg, and C. Ebeling, "Specifying and compiling applications for rapid," In *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines, 15-17 April 1998*, pp. 116–25, Dept. of Comput. Sci. & Eng., Washington Univ., Seattle, WA, USA, IEEE Comput. Soc, Napa Valley, CA, USA, / 1998.

[Cron99] D. C. Cronquist, C. Fisher, M. Figueroa, P. Franklin, and C. Ebeling, "Architecture design of reconfigurable pipelined datapaths," In *Proceedings 20th Anniversary Conference on Advanced Research in VLSI, 21-24 March 1999*, pp. 23–40, Dept. of Comput. Sci. & Eng., Washington Univ., Seattle, WA, USA, IEEE Comput. Soc, Atlanta, GA, USA, / 1999.

[Deb02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, No. 2, pp. 182–97, 04/ 2002.

[Dont03] S. Donthi and R. L. Haggard, "A survey of dynamically reconfigurable fpga devices," In *Proceedings of the 35th Southeastern Symposium on System Theory, 16-18 March 2003*, pp. 422–6, Dept. of Electr. & Comput. Eng., Tennessee Technol. Univ., Cookeville, TN, USA, IEEE, Morgantown, WV, USA, / 2003.

[Ebel96] C. Ebeling, D. C. Cronquist, and P. Franklin, "Rapid-reconfigurable pipelined datapath," In *6th International Workshop on Field-Programmable Logic and Applications. FPL '96, 23-25 Sept. 1996*, pp. 126–35, Dept. of Comput. Sci. & Eng., Washington Univ., Seattle, WA, USA, Springer-Verlag, Darmstadt, Germany, / 1996.

[Ebel97] C. Ebeling, D. C. Cronquist, P. Franklin, J. Secosky, and S. G. Berg, "Mapping applications to the rapid configurable architecture," In *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No.97TB100186), 16-18 April 1997*, pp. 106–15, Dept. of Comput. Sci. & Eng., Washington Univ., Seattle, WA, USA, IEEE Comput. Soc, Napa Valley, CA, USA, / 1997.

[Erba03] C. Erbas, S. C. Erbas, and A. D. Pimentel, "A multiobjective optimization model for exploring multiprocessor mappings of process networks," In *First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis, 1-3 Oct. 2003*, pp. 182–7, Dept. of Comput. Sci., Amsterdam Univ., Netherlands, ACM, Newport Beach, CA, USA, / 2003.

[Erba06] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design," *IEEE Transactions on Evolutionary Computation*, vol. 10, No. 3, pp. 358–74, 06/ 2006.

[Fons93] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: formulation, discussion and generalization," In *Proceedings of ICGA-93: Fifth International Conference on Genetic Algorithms, 17-22 July 1993*, pp. 416–23, Dept. Automatic Control & Syst. Eng., Sheffield Univ., UK, Morgan Kaufmann, Urbana-Champaign, IL, USA, / 1993.

[Forn02] William Fornaciari, Donatella Sciuto, Cristina Silvano, and Vittorio Zaccaria, "A sensitivity-based design space exploration methodology for embedded systems," *Design Automation for Embedded Systems*, vol. 7, No. 1-2, pp. 7–33, 2002.

[Giva02a] T. Givargis and F. Vahid, "Platune: a tuning framework for system-on-a-chip platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, No. 11, pp. 1317–27, 11/ 2002.

[Giva02b] Tony Givargis, Frank Vahid, and Jorg Henkel, "System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip (december 2002)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, No. 4, pp. 416–422, 2002.

[GmbH] PACT Informationstechnologie GmbH, "The xpp white paper," .

[Gokh05] Maya B. Gokhale and Paul S. Graham, *Reconfigurable Computing*, Springer, 2005.

[Gold00] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "Piperench: a reconfigurable architecture and compiler," *Computer*, vol. 33, No. 4, pp. 70–7, 04/ 2000.

[Govi05] S. Govind and R. Govindarajan, "Performance modeling and architecture exploration of network processors," In *Proceedings. Second International*

Conference on the Quantitative Evaluation of Systems, 19-22 Sept. 2005, pp. 189–98, Supercomput. Educ. & Res. Center, Indian Inst. of Sci., Bangalore, India, IEEE Comput. Soc, Torino, Italy, / 2005.

[Grie04] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integration, The VLSI Journal*, vol. 38, No. 2, pp. 131–83, 12/ 2004.

[Hadj00] G. Hadjiyiannis, S. Hanono, and S. Devadas, "Isdl: an instruction set description language for retargetability and architecture exploration," *Design Automation for Embedded Systems*, vol. 6, No. 1, pp. 39–69, 2000.

[Hadj03] G. Hadjiyiannis and S. Devadas, "Techniques for accurate performance evaluation in architecture exploration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, No. 4, pp. 601–15, 08/ 2003.

[Hadj99] G. Hadjiyiannis, P. Russo, and S. Devadas, "A methodology for accurate performance evaluation in architecture exploration," In *Proceedings 1999 Design Automation Conference, 21-25 June 1999*, pp. 927–32, Lab. for Comput. Sci., MIT, Cambridge, MA, USA, IEEE, New Orleans, LA, USA, / 1999.

[Hart01] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," In *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001, 13-16 March 2001*, pp. 642–9, Dept. of Comput. Sci., Kaiserslautern Univ., Germany, IEEE Comput. Soc, Munich, Germany, / 2001.

[Hart94] R. W. Hartenstein, R. Kress, and H. Reinig, "A new fpga architecture for word-oriented datapaths," In *Field-Programmable Logic. Architectures, Synthesis and Applications. 4th International Workshop on Field-Programmable Logic and Applications, FPL '94. Proceedings, 7-9 Sept. 1994*, pp. 144–55, Kaiserslautern Univ., Germany, Springer-Verlag, Prague, Czech Republic, / 1994.

[Haub05] C. Haubelt, S. Otto, C. Grabbe, and J. Teich, "A system-level approach to hardware reconfigurable systems," In *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference 2005, 18-21 Jan. 2005*, pp. 298–301, Dept. of Comput. Sci., Erlangen-Nuremberg Univ., Erlangen, Germany, IEEE, Shanghai, China, / 2005.

[Haus97] J. R. Hauser and J. Wawrzynek, "Garp: a mips processor with a reconfigurable coprocessor," In *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No.97TB100186), 16-18 April 1997*, pp. 12–21, California Univ., Berkeley, CA, USA, IEEE Comput. Soc, Napa Valley, CA, USA, / 1997.

[Helg03] D. R. Helgemo, "Digital signal processing at 1 ghz in a field-programmable object array," In *IEEE International SOC Conference, 17-20 Sept. 2003*, pp. 57–60, MathStar, Inc., Minneapolis, MN, USA, IEEE, Portland, OR, USA, / 2003.

[Hill93] M. D. Hill, J. R. Larus, A. R. Lebeck, M. Talluri, and D. A. Wood, “Wisconsin architectural research tool set,” *Computer Architecture News*, vol. 21, No. 4, pp. 8–10, 1993.

[Horn94] J. Horn, N. Nafpliotis, and D. E. Goldberg, “A niched pareto genetic algorithm for multiobjective optimization,” In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, 27-29 June 1994*, pp. 82–7, Genetic Algorithms Lab., Illinois Univ., Urbana, IL, USA, IEEE, Orlando, FL, USA, / 1994.

[Josh06] A. Joshi, J. J. Yi, R. H. Bell Jr, L. Eeckhout, L. John, and D. Lilja, “Evaluating the efficacy of statistical simulation for design space exploration,” In *ISPASS 2006. IEEE International Symposium on Performance Analysis of Systems Software, 19-21 March 2006*, pp. 70–9, Dept. of Electr. & Comput. Eng., Texas Univ., Austin, TX, USA, IEEE, Austin, TX, USA, / 2006.

[Kahn74] Gilles Kahn, “The semantics of a simple language for parallel programming,” *Proc. of the IFIP Congress 74*, pp. 471–475, 1974.

[Keut00] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, “System-level design: orthogonalization of concerns and platform-based design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, No. 12, pp. 1523–43, 12/ 2000.

[Khar01] A. Khare, A. Halambi, N. Savoiu, P. Grun, N. Dutt, and A. Nicolau, “V-sat: A visual specification and analysis tool for system-on-chip exploration,” *Journal of Systems Architecture*, vol. 47, No. 3-4, pp. 263–75, 04/ 2001.

[Kim03] Sungchan Kim, Chaeseok Im, and Soonhoi Ha, “Schedule-aware performance estimation of communication architecture for efficient design space exploration,” In *First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis, 1-3 Oct. 2003*, pp. 195–200, Sch. of Eng. & Comput. Sci., Seoul Nat. Univ., South Korea, ACM, Newport Beach, CA, USA, / 2003.

[Kim04] M. Kim, T. Hiroyasu, M. Miki, and S. Watanabe, “Spea2+: improving the performance of the strength pareto evolutionary algorithm 2,” In *Parallel Problem Solving from Nature - PPSN VIII. 8th International Conference. Proceedings, 18-22 Sept. 2004*, pp. 742–51, Dept. of Knowledge Eng. & Comput. Sci., Doshisha Univ., Kyoto, Japan, Springer-Verlag, Birmingham, UK, / 2004.

[Kim06] Sungchan Kim and Soonhoi Ha, “Efficient exploration of bus-based system-on-chip architectures,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, No. 7, pp. 681–92, 07/ 2006.

[Know99] J. Knowles and D. Corne, “The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation,” In *Proceedings of the 1999. Congress on Evolutionary Computation-CEC99, 6-9 July 1999,* pp. 98–105, Dept. of Comput. Sci., Reading Univ., UK, IEEE, Washington, DC, USA, / 1999.

[Kris06] V. Krishnan and S. Katkoori, “A genetic algorithm for the design space exploration of datapaths during high-level synthesis,” *IEEE Transactions on Evolutionary Computation*, vol. 10, No. 3, pp. 213–29, 06/ 2006.

[Labo06] George Mason University’s ECLab Evolutionary Computation Laboratory, “Ecj release 14 and 15,” 2006.

[Lahi04] K. Lahiri, A. Raghunathan, and S. Dey, “Design space exploration for optimizing on-chip communication architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, No. 6, pp. 952–61, 06/ 2004.

[Le03] Edward A. Le, “Overview of the ptolemy project,” Technical Report UCB/ERL M03/25, University of California, Berkeley, CA, 94720, USA,, July 2003.

[Liev01] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere, “A methodology for architecture exploration of heterogeneous signal processing systems,” In *1999 IEEE Workshop on Signal Processing Systems. SiPS 99. Design and Implementation, 20-22 Oct. 1999*, pp. 197–207, Dept. of Inf. Technol. & Syst., Delft Univ. of Technol., Netherlands, Kluwer Academic Publishers, Taipei, Taiwan, 11/ 2001.

[LN03] Katarzyna Leijten-Nowak and Jef L. Van Meerbergen, “An fpga architecture with enhanced datapath functionality,” In *ACM/SIGDA 11th ACM International Symposium on Field Programmable Gate Arrays, Feb 23-25 2003*, pp. 195–204, Eindhoven University of Technology, Design Automation Section, Eindhoven, Netherlands, Association for Computing Machinery, Monterey, CA, United States, 2003 Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[Magn95] P. Magnusson and B. Werner, “Efficient memory simulation in simics,” In *Proceedings of Simulation Symposium, 9-13 April 1995*, pp. 62–73, Swedish Inst. of Comput. Sci., Kista, Sweden, IEEE Comput. Soc. Press, Phoenix, AZ, USA, / 1995.

[Mars99] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, “A reconfigurable arithmetic array for multimedia applications,” In *Proceedings of FPGA ’99. ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays, 21-23 Feb. 1999*, pp. 135–43, Hewlett Packard Labs., Bristol, UK, ACM, Monterey, CA, USA, / 1999.

[Mart02] Grant Martin and Jean-Yves Brunel, “Platform-based co-design and co-development: Experience, methodology and trends,” In *The ninth IEEE/DATC Electronic Design Processes Workshop*, April 2002.

[Matt04] Daniel Mattsson and Marcus Christensson, *Evaluation of Synthesizable CPU cores* Master's thesis, Computer Science and Engineering Program, Department of Computer Engineering,Chalmers University of Technology, 2004.

[Mira05] B. Miramond and J. M Delosme, "Design space exploration for dynamically reconfigurable architectures," In *Proceedings. Design, Automation and Test in Europe, 7-11 March 2005*, pp. 366–71, LaMI, Univ. d'Evry Val d'Essonne, France, IEEE Comput. Soc, Munich, Germany, / 2005.

[Mirs96] E. Mirsky and A. DeHon, "Matrix: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," In *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines, 17-19 April 1996*, pp. 157–66, IEEE Comput. Soc. Press, Napa Valley, CA, USA, / 1996.

[Mish03] P. Mishra, A. Kejariwal, and N. Dutt, "Rapid exploration of pipelined processors through automatic generation of synthesizable rtl models," In *Proceedings 14th IEEE International Workshop on Rapid Systems Prototyping, 9-11 June 2003*, pp. 226–32, Architectures & Compilers for Embedded Syst. Lab., California Univ., Irvine, CA, USA, IEEE Comput. Soc, San Diego, CA, USA, / 2003.

[Mitc96] Melanie Mitchell, *An Introduction To Genetic Algorithms*, The MIT Press, 1996.

[Miya98] T. Miyamori and U. Olukotun, "A quantitative analysis of reconfigurable coprocessors for multimedia applications," In *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines, 15-17 April 1998*, pp. 2–11, Syst. ULSI Eng. Lab., Toshiba Corp., Japan, IEEE Comput. Soc, Napa Valley, CA, USA, / 1998.

[Moha02] S. Mohanty and V. K. Prasanna, "Rapid system-level performance evaluation and optimization for application mapping onto soc architectures," In *Proceedings 15th Annual IEEE International ASIC/SOC Conference, 25-28 Sept. 2002*, pp. 160–7, Dept. of Electr. Eng., Univ. of Southern California, Los Angeles, CA, USA, IEEE, Rochester, NY, USA, / 2002.

[Mouh06] R. B. Mouhoub and O. Hammami, "Multiprocessor on chip: beating the simulation wall through multiobjective design space exploration with direct execution," In *Proceedings. 20th International Parallel and Distributed Processing Symposium, 25-29 April 2006*, pp. 8, ENSTA, France, IEEE, Rhodes Island, Greece, / 2006.

[Niar06] S. Niar and N. Inglart, "Rapid performance and power consumption estimation methods for embedded system design," In *Proceedings Seventeenth IEEE International Workshop on Rapid System Prototyping, 14-16 June 2006*, pp. 47–53, Universite de Valenciennes et du Hainaut-Cambresis, France, IEEE Comput. Soc, Chania, Crete, Greece, / 2006.

[Oski00] M. Oskin, F. T. Chong, and M. Farrens, “Hls: combining statistical and symbolic simulation to guide microprocessor designs,” In *Proceedings of 27th International Symposium on Computer Architecture, 10-14 June 2000*, pp. 71–82, Dept. of Comput. Sci., California Univ., Davis, CA, USA, ACM, Vancouver, BC, Canada, / 2000 Also available on CD-ROM in PDF format.

[Oyam04] M. S. Oyamada, F. Zschornack, and F. R. Wanger, “Accurate software performance estimation using domain classification and neural networks,” In *Proceedings. SBCCI 2004. 17th Symposium on Integrated Circuits and Systems Design, 7-11 Sept. 2004*, pp. 175–80, Inst. de Inf., Univ. Fed. do Rio Grande do Sul, Porto Alegre, Brazil, ACM, Porto de Galinhas, Pernambuco, Brazil, / 2004.

[Pale03] G. Palermo, C. Silvano, and V. Zaccaria, “A flexible framework for fast multi-objective design space exploration of embedded systems,” *Integrated Circuit and System Design*, vol. 2799, pp. 249–258, 2003.

[Pale04] C. Palermo and C. Silvano, “Pirate: A framework for power/performance exploration of network-on-chip architectures,” *Integrated Circuit and System Design*, vol. 3254, pp. 521–531, 2004.

[Pare96] V. Pareto, “Cours d’economie politique,” F. Rouge, Lausanne, Technical Report, F. Rouge, Lausanne, 1896.

[Pime01] A. D. Pimentel, L. O. Hertzbetger, P. Lieverse, P. van der Wolf, and E. E. Deprettere, “Exploring embedded-systems architectures with artemis,” *Computer*, vol. 34, No. 11, pp. 57–63, 11/ 2001.

[Pime06] A. D. Pimentel, C. Erbas, and S. Polstra, “A systematic approach to exploring embedded system architectures at multiple abstraction levels,” *IEEE Transactions on Computers*, vol. 55, No. 2, pp. 99–112, FEB 2006.

[Rudo01] ”Günter Rudolph”, “”evolutionary search under partially ordered finite sets”,” In ”M F. Sebaaly”, editor, *”Proceedings of the International NAISO Congress on Information Science Innovations (ISI 2001)”*, pp. ”818–822”, ”ICSC Academic Press”, ”2001”.

[Schm02] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. Reed Taylor, “Piperench: A virtualized programmable datapath in 0.18 micron technology,” In *Proceedings of the IEEE 2002 Custom Integrated Circuits Conference, 12-15 May 2002*, pp. 63–6, Dept. of Electr. & Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, USA, IEEE, Orlando, FL, USA, / 2002 Also available on CD-ROM in PDF format.

[Shin04] Chulho Shin, Young-Taek Kim, Eui-Young Chung, Kyu-Myung Choi, Jeong-Taek Kong, and Soo-Kwan Eo, “Fast exploration of parameterized bus architecture for communication- centric soc design,” In *Proceedings - Design, Automation and Test in Europe Conference and Exhibition, DATE*

04, Feb 16-20 2004, pp. 352–357, Institute of Electrical and Electronics Engineers Computer Society, Piscataway, United States, Paris, France, 2004 Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[Slom04] Frank Slomka, Karsten Albers, and Richard Hofmann, “A multiobjective tabu search algorithm for the design space exploration of embedded systems.,” In *DIPES*, pp. 227–236, 2004.

[Srik00] S. Srikanteswara, J. H. Reed, P. Athanas, and R. Boyle, “A soft radio architecture for reconfigurable platforms,” *IEEE Communications Magazine*, vol. 38, No. 2, pp. 140–7, 02/ 2000.

[Srin94] ”N Srinivas and Kalyanmoy Deb”, “”multiobjective optimization using nondominated sorting in genetic algorithms”,” *”Evolutionary Computation”*, vol. ”2”, No. ”3”, pp. ”221–248”, ”1994”.

[Suma02] B. Suman, “Multiobjective simulated annealing - a metaheuristic technique for multiobjective optimization of a constrained problem,” *Foundations of Computing and Decision Sciences*, vol. 27, No. 3, pp. 171–91, / 2002.

[Suma04] B. Suman, “Study of simulated annealing based algorithms for multiobjective optimization of a constrained problem,” *Computers & Chemical Engineering*, vol. 28, No. 9, pp. 1849–71, 08/15 2004.

[SV04] A. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis, and M. Sgroi, “Benefits and challenges for platform-based design,” In *Proceedings 2004. Design Automation Conference, 7-11 June 2004*, pp. 409–14, Dept. of EECS, California Univ., Berkeley, CA, USA, ACM, San Diego, CA, USA, / 2004.

[Tayl02] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Jae-Wook Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, “The raw microprocessor: a computational fabric for software circuits and general-purpose programs,” *IEEE Micro*, vol. 22, No. 2, pp. 25–35, 03/ 2002.

[Vahi95] F. Vahid and D. D. Gajski, “Slif: a specification-level intermediate format for system design,” In *Proceedings the European Design and Test Conference. ED&TC 1995, 6-9 March 1995*, pp. 185–9, Dept. of Comput. Sci., California Univ., Riverside, CA, USA, IEEE Comput. Soc. Press, Paris, France, / 1995.

[vdH00] P. van den Hamer, W. P. M. van der Linden, P. Bingley, and N. W. Schellingerhout, “A system simulation framework. software environments for designing complex products,” In *Proceedings of ACM/IEEE-CAS/EDAC Design Automation Conference, 5-9 June 2000*, pp. 699–704, Philips Res. Lab., Eindhoven, Netherlands, ACM, Los Angeles, CA, USA, / 2000.

[Vija00] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye, “Energy-driven integrated hardware-software optimizations using simplepower,” In

ISCA-27: The 27th Annual International Symposium on Computer Architecture, Jun 10-Jun 14 2000, pp. 95–106, Pennsylvania State Univ, University Park, PA, USA, Institute of Electrical and Electronics Engineers Computer Society, Los Alamitos, CA, USA, Vancouver, BC, Can, 2000 Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[Wain97] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal, "Baring it all to software: Raw machines," *Computer*, vol. 30, No. 9, pp. 86–93, 1997.

[Wang05] Haili Wang, Jinian Bian, Yawen Niu, Kun Tong, and Yunfeng Wang, "Ca-ex: a tuning-incremental methodology for communication architectures in embedded systems," In *Embedded Software and Systems. First International Conference, ICESS 2004. Revised Selected Papers, 9-10 Dec. 2004*, pp. 74–80, Dept. of Comput. Sci. & Technol., Tsinghua Univ., Beijing, China, Springer-Verlag, Hangzhou, China, / 2005.

[Wang93] Q. Wang and P. G. Gulak, "An array architecture for reconfigurable datapaths," In W. Moore and W. Luk, editors, *More FPGAs. Oxford International Workshop on Field-Programmable Logic and Applications*, pp. 35–46, Abingdon EE&CS Books, Oxford, England, 1993.

[Xili04] Inc Xilinx, "Two flows for partial reconfiguration: Module based or difference based," 2004.

[Xili06] Inc Xilinx, "Virtex-5 family overview," October 2006.

[Yi06] J. J. Yi and D. J. Lilja, "Simulation of computer architectures: simulators, benchmarks, methodologies, and recommendations," *IEEE Transactions on Computers*, vol. 55, No. 3, pp. 268–80, 03/ 2006.

[Zabe06] M. Zabel, S. Kohler, M. Zimmerling, T. B. Preuber, and R. G. Spallek, "Design space exploration of coarse-grain reconfigurable dsps," In *2005 International Conference on Reconfigurable Computing and FPGAs ReConFig 2005, 28-30 Sept. 2005*, pp. 8, Inst. of Comput. Eng., Dresden Univ. of Technol., Germany, IEEE Computer Society, Puebla City, Mexico, / 2006.

[Zhan00] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey, "A 1-v heterogeneous reconfigurable dsp ic for wireless baseband digital signal processing," In *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, 7-9 Feb. 2000*, pp. 1697–704, Dept. of Electr. Eng. & Comput. Sci., California Univ., Berkeley, CA, USA, IEEE, San Francisco, CA, USA, 11/ 2000.

[Zitz01] "Eckart Zitzler, Marco Laumanns, and Lothar Thiele", ""spea2: Improving the strength pareto evolutionary algorithm"," -, Technical Report, "Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich", "May" 2001.

[Zitz02] Eckart Zitzler, Marco Laumanns, and Lothar Thiele, “Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization,” In K. C. Giannakoglou, D. T. Tsahalis, J. Periaux, K. D. Papaliliou, and T. Fogarty, editors, *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems. Proceedings of the EUROGEN2001 Conference, Athens, Greece, September 19-21, 2001*, pp. 95–100, International Center for Numerical Methos in Engineering, CIMNE, 2002.

[Zitz99] E. Zitzler and L. Thiele, “Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach,” *IEEE Transactions on Evolutionary Computation*, vol. 3, No. 4, pp. 257–71, 11/ 1999.

[Zivk03] V. D. Zivkovic, E. Deprettere, E. de Kock, and P. van der Wolf, “Fast and accurate multiprocessor architecture exploration with symbolic programs,” In *6th Design Automation and Test in Europe (DATE 03), 3-7 March 2003*, pp. 656–61, Leiden Univ., Netherlands, IEEE Comput. Soc, Munich, Germany, / 2003.