Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 54074, Pages 1–14
DOI 10.1155/ES/2006/54074

MOCDEX: Multiprocessor on Chip Multiobjective Design Space Exploration with Direct Execution

Riad Ben Mouhoub and Omar Hammami

UEI, ENSTA 32, Boulevard Victor, 75739 Paris, France

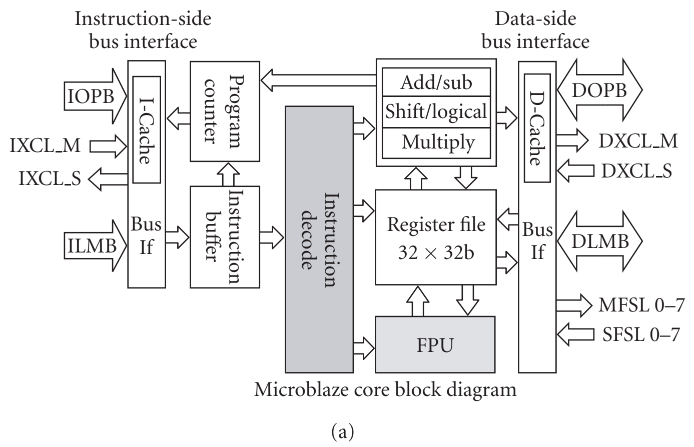
Received 15 December 2005; Revised 5 May 2006; Accepted 2 June 2006

Fully integrated system level design space exploration methodologies are essential to guarantee efficiency of future large scale system on programmable chip. Each design step in the design flow from system architecture to place and route represents an optimization problem. So far, different tools (computer architecture, design automation) are used to address each problem separately with at best estimation techniques from one level to another. This approach ignores the various and very diverse vertical relations between distinct levels parameters and provides at best local optimization solutions at each step. Due to the large scale of SoC, system level design methodologies need to tackle the system design process as a global optimization problem by fully integrating physical design in the design space exploration. We propose MOCDEX, a multiobjective design space exploration methodology, for multiprocessor on chip which closes the gap between these associated tools in a fully integrated approach and with hardware in the loop. A case study of a 4-way multiprocessor demonstrates the validity of our approach.

1. INTRODUCTION

System on chip are increasingly becoming complex to design, test, and fabricate. SoC design methodologies make intensive use of intellectual properties (IPs) [1] to reduce the design cycle time and meet stringent time to market constraints. However, associated tools still lag behind when addressing the huge associated design space exposed by the combination of soft IP. In addition, failure to meet an efficient distribution in terms of performance, area, and energy consumption makes the whole design inappropriate. Although this problem is already hard to solve in the ASIC domain, it is exacerbated in the system on programmable chip (SoPC) domain. SoPC are large scale devices offering abundant resources but in fixed amount and in fixed location on chip. Implementing embedded multiprocessors on these devices presents several advantages, the most important is to be able to quickly evaluate various configurations and tune them accordingly. Indeed, embedded multiprocessor design is highly application-driven and it is therefore highly advantageous to execute applications on real prototypes. However, due to the fact that specific resources are located at fixed positions on these large chips it is hard not to take into account the important impact of place and route results on the critical paths and therefore on the overall performance. In this paper, we address this multiobjective optimization problem [2] restricted to performance and area through the combination of an efficient design space exploration (DSE) technique coupled with direct execution on an FPGA board [3]. The direct execution removes the prohibitive simulation time associated with the evaluation of embedded multiprocessor systems. A side effect of this approach is that direct execution requires actual on chip implementation of the various multiprocessor configurations to be explored which provides actual post synthesis and place and route area information. The resulting flow is fully integrated from multiprocessor platform specification to execution.

The paper is organized as follows. In Section 2, we review previous work. Section 3 describes an example of soft IP-based multiprocessor and the breadth of the problem associated with the design of such multiprocessor on a particular instance of embedded memories optimization. Section 4 presents our approach, MOCDEX, based on multiobjective evolutionary algorithms (EA) and direct execution. In Section 5 we describe a case study and validation, while Section 6 provides exploration results. Section 7 provides statistical insight in the explored design space and demonstrates the diversity of multiprocessor configurations explored during the automatic process. Finally, we conclude in Section 8 with remarks and directions for future work.



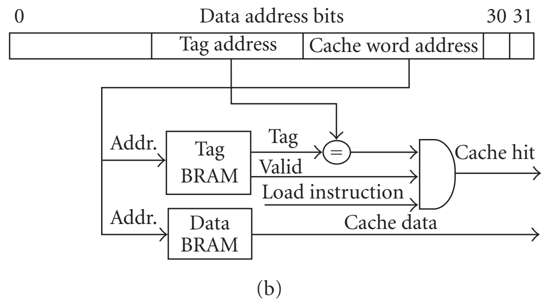


Figure 1: (a) MicroBlaze soft IP processor. (b) MicroBlaze processor cache organization.

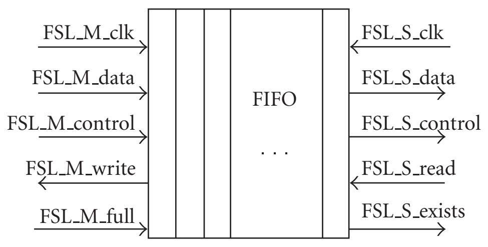


Figure 2: Fast simplex link.

2. PREVIOUS WORK

The recent emergence of multiprocessors on chip as strong potential candidates to address performance, energy, and area constraints for embedded applications has resulted in the following question: how do we design efficient multiprocessors on chip for a target application? Design automation tools fail to address this question, while traditional parallel computer architectures techniques [4] have not been exposed to the huge diversity brought by soft IP-based design methodologies and the strong constraints of embedded systems [5]. Therefore, the design of multiprocessor on chip is the convergence focus of previously unrelated techniques and as such represents a new problem on how to establish a close integration between those techniques. It is then not surprising that few works so far have been devoted to design methodologies for multiprocessors on chip. In [6] they present a design flow for the generation of application-specific multiprocessor architectures. In the flow, architectural parameters are first extracted from a high-level specification and are used to instantiate architectural components such as processors, memory modules, and communication networks. Cycle accurate cosimulations of the architectures are used for performance evaluation while all results in our case are obtained through actual execution and they do not use design space exploration algorithm. In [7], synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors is proposed based on an iterative improvement algorithm implemented in the context of a commercial design flow. The proposed algorithm is based on cycle count estimation and instruction-set simulations, and although synthesis results are used, both architecture and implementation flows are still decoupled. In [8] they propose an automated exploration framework for FPGA-based soft multiprocessor systems. Using as input the application graph that describes tasks and communication links, outputs of the exploration step are a microarchitecture configuration of processors and communication channels, a mapping of the application tasks and links onto the processors and channels of the micro-architecture. They formulate the exploration problem as an integer linear problem. The “best design” based on the ILP results is selected and synthesized to verify performance. This verification may fail because routing details are not taken into account during the exploration process. This approach still keeps decoupled design automation tools and exploration, while in our approach design space exploration fully integrates design automation tools since solutions are ranked on the area results obtained post-synthesis and place and route and performance results obtained from actual execution on board. Besides, the problem formulation ignores the arbitration overhead when computing the communication access time again due to the static nature of the design space exploration decoupled from actual execution. As pointed out by the authors, this can lead to a significant source of errors when there are a large number of masters on the bus. Finally, it should be clear that no single “best design” exists in any multiobjective optimization problem and only a Pareto set can be obtained. In [9] they present high-level scheduling and interconnect topology synthesis techniques for embedded multiprocessor system-on-chip that are streamlined for one or more digital signal processing applications. The proposed interconnect synthesis method utilizes a genetic algorithm (GA) operating in conjunction with a list scheduling algorithm which produces candidate topology graphs based on direct physical communication. The proposed algorithm is a single objective algorithm, while the algorithm used in our work is a multiobjective algorithm; and although we use direct link we optimize also buffering capacities by trading on-chip memory among embedded processor cache memories and connection link buffers. To the best of our knowledge our work is the first to fully integrate and therefore close the gap between design automation tools and architecture design space exploration technique in a multiobjective constraints paradigm with actual execution for all multiprocessor on chip configurations explored during the design space exploration process.

3. SOFT IP-BASED EMBEDDED MULTIPROCESSOR SYSTEMS

Soft IP-based embedded multiprocessor systems are SoC fully designed with soft IPs. This includes soft IP processors, interconnect infrastructure and memories. An example of such soft IP multiprocessor is described below based on Xilinx EDK IPs [10].

3.1. *MicroBlaze soft IP processor*

MicroBlaze soft IP [11] is a 32-bit 3-stage single issue pipelined Harvard style embedded processor architecture provided by Xilinx as part of their embedded design tool kit.

Both caches are direct mapped, with 4-word cache lines allowing configurable cache and tag size and user selectable cacheable memory area. Data cache uses a write-through policy. MicroBlaze core configurability extends to functional unit through user selectable barrel shifter (BS), hardware multiplier (HWM), hardware divider (HWD), and floating point unit (FPU). MicroBlaze has neither static nor dynamic branch prediction unit and supports branches with delay slots. For its communication purposes, MicroBlaze uses either a bus or a direct link. The on-chip peripheral bus (OPB) is part of IBM CoreConnect bus architecture and allows the design of complete single processor systems with peripherals and uses designed hardware accelerators [12, 13]. However, even for a simple embedded-processor-based multiprocessors designs such as MicroBlaze, the OPB bus is not suitable because of its lack of scalability. Another approach is provided by “Fast Simplex Link” [14] which allows direct connection between embedded processors through FIFO channels.

3.2. *MicroBlaze fast simplex link*

The fast simplex link (FSL) [14] is an IP developed by Xilinx to achieve a fast unidirectional point-to-point communication between any two components. The FSL link is implemented as a 32-bit wide FIFO with configurable depth and width option. The FSL can be either a master or a slave interface depending upon its use.

MicroBlaze soft embedded processor allows up to 8 master and slave FSL interfaces. Basic software drivers are provided to simplify the use of FSL connection. They consist of read/write routines and control functions. The read/write routines can be executed in two different ways: blocking and nonblocking mechanism.

3.3. *IBM interconnect*

The IBM interconnect [10] represents a set of IPs used to develop SoC devices. It includes the PLB and OPB bus, a PLB-OPB bridge, and various peripherals.

3.4. *MPSoC platform description*

Our FPGA multiprocessor platform consists of four MicroBlaze processors with instruction and data cache units. These processors are connected with each other through FSL channels.

Each MicroBlaze is connected, as shown in Figure 3, to an OPB bus to use a timer and an interrupt controller for threads and OS execution. MicroBlaze MB0 is connected to the OPB bus which is connected to the PCI interface of the host (WS). This allows the designer to send and receive data from the host to the multiprocessor system. We implemented a soft layer of communication in each MicroBlaze which performs send and receive functions of packets. The packets consist of headers representing the destination and source addresses and the number of flits in the payload. A wormhole routing algorithm was used since it uses less memory, making it suitable for network on chip communication. As it can be seen a 4-way multiprocessor has been built based on the previously described soft IPs.

The implementation of such a soft IP multiprocessor on FPGA platform requires a variable amount of resources as each soft IP composing the multiprocessor requires a variable amount of resources depending on the configuration options [10]. Table 1 provides an insight on such variability.

Such a soft IP multiprocessor can be easily adapted to the need of a specific application adapted to a particular application. However, these systems for best efficiency and low memory latency require the use of embedded on chip memories. Unfortunately, embedded memories are scarce resources for which processors instruction and data cache memories as well as bus and network on-chip FIFO-based interfaces will compete. This competition is dominated by the absolute requirement of efficiency in performance, area, and energy consumption [5]. If we focus on cache and FSL configurability, we have for each cache memory 7 possible configurations and for the FSL 11 possible configurations. The design space associated with those parameters ($7^4 \times 11^8$, thus 514 675 673 281 different configurations) requires 16 321 years of simulation for 1 minute simulation per configuration.

4. MOCDEX MULTIOBJECTIVE DESIGN SPACE EXPLORATION

4.1. *Problem formulation*

The design challenge represented by soft IP-based multiprocessor design is a multiobjective optimization problem [2].

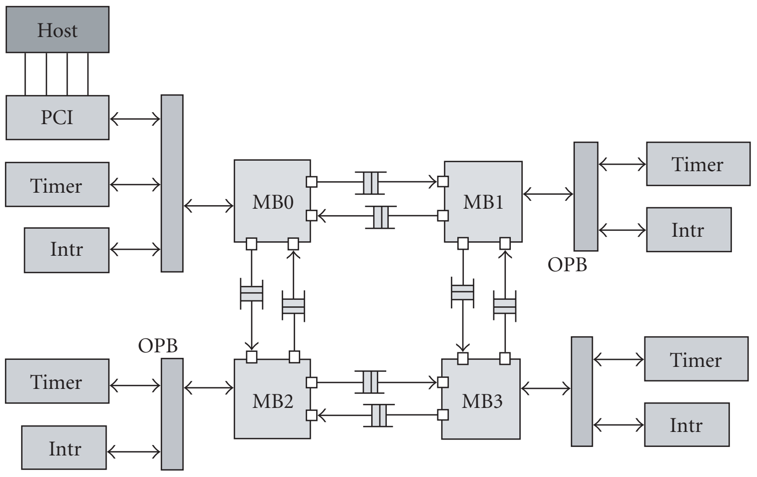


Figure 3: Mesh platform 2×2.

Table 1: Multiprocessor soft IP resources variation.

Soft IP	Slices Min Max	FF Min Max	BRAM Min Max	Parameters	Soft IP	Slices Min Max	FF Min Max	BRAM Min Max	Parameters
MicroBlaze	731 var	552 var	0 var	Cache sizes 1 K, 2 K, 4 K, 8 K, 16 K, 32 K, 64 K	OPB	46 410	5 121	N/A N/A	Data bus width, address bus width, arbiter
					OPB PCI	340 3025	445 2105	0 2+	Interface/DMA parameters
FSL width/depth	21 451	36 34	0 17	FIFO sizes 8, 16, 32, 64, 128, 256, 512, 1 K, 2 K, 4 K, 8 K	OPB timer	99 200	105 266	0	Timer counter widths
					OPB intr ctr	54 307	63 342	0	Number of interrupt inputs

The multiobjective optimization problem is the problem of simultaneously minimizing the n components (e.g., area, number of execution cycles, energy consumption), f_k, $k = 1, \ldots, n$, of a possibly nonlinear function f of a general decision variable x in a universe U, where

$$f(x) = (f_1(x), f_2(x), \ldots, f_n(x)). \quad (1)$$

The problem has usually no unique optimal solution but a set of nondominated alternative solutions known as the Pareto-optimal set. The dominance is defined as follows.

Definition 1 (Pareto dominance). A given vector $u = (u_1, u_2, \ldots, u_n)$ is said to dominate $v = (v_1, \ldots, v_n)$ if and only if u is partially less than v ($u_p < v$), that is,

$$\forall i \in \{1, \ldots, n\}, \quad u_i \leq v_i, \quad \exists i \in \{1, \ldots, n\} : u_i < v_i. \quad (2)$$

The Pareto optimality definition derives from the Pareto dominance.

Definition 2 (Pareto optimality). A solution $x_u \in U$ is said to be Pareto optimal if and only if there is no $x_v \in U$ for which $v = f(x_v) = (v_1, \ldots, v_n)$ *dominates* $u = f(x_u) = (u_1, \ldots, u_n)$.

Pareto-optimal solutions are also called efficient, nondominated, and noninferior solutions. The corresponding objective vectors are simply called nondominated. The set of all nondominated vectors is known as the nondominated set or the Pareto set (also Pareto-optimal set or Pareto-optimal front). This Pareto set can be seen as the tradeoff surface of the problem. The solution of a practical problem such as multiprocessor system on chip (MPSoC) design may be constrained by a number of restrictions imposed on a decision variable. Constraints may express the domain of definition of the objective function or alternatively impose further restrictions on the solution of the problem according to knowledge at a higher level. In the general case of system on programmable chip, the amount of on chip memory for example is fixed and represents a clear and stringent constraint. The constrained optimization problem is that of minimizing a multiobjective function $(f_1, \ldots, f_k)$ of some generic decision variable x in a universe U subject to a positive number $n - k$ of conditions involving x and eventually expressed as a functional vector inequality of the type

$$(f_{k+1}(x), \ldots, f_n(x)) < (g_{k+1}, \ldots, g_n), \tag{3}$$

where the inequality applies component-wise. It is implicitly assumed that there is at least one point in U which satisfies all constraints although in practice that cannot always be guaranteed.

The case study of multiobjective optimization we will address in this paper is the minimization of area (BRAM $f1$ and slices resources $f2$) and execution time (number of cycles $f3$) representing a 3-objectives multiobjective problem.

4.2. Multiobjective optimization and multiobjective evolutionary algorithms (MOEA)

Multiobjective optimization have not been addressed properly by traditional optimization techniques (gradient based, simulated annealing, linear programing) since most of these techniques are mono-objective. Extending these techniques through approaches using aggregation functions does not represent true multiobjective optimization and does not produce multiple solutions. Multiobjective evolutionary algorithms (MOEA) are more appropriate to solve optimization problems with concurrent conflicting objectives and are particularly suited for producing Pareto-optimal solutions. Several Pareto-based evolutionary algorithms have been proposed during the last decade, SPEA-2, PESA, and NSGA-II, [2, 15] to solve multicriteria optimization problems. The NSGA-II [16] is an MOEA considered to outperform other MOEA [17] and is briefly presented below.

Individuals classification

Initially, before carrying out the selection, one assigns to each individual in the population a row *rank* (by using the Pareto set). All the nondominated individuals of the same row are classified in a category. To this category, we assign effectiveness, which is inversely proportional to the order of Pareto set. Figure 4 presents an example of classification in Pareto sets.

Main loop of algorithm NSGA-II [16]

Initially, a random parent population $P0$ is created. Each individual of this population is affected to an adequate Pareto rank. From the population $P0$, we apply the genetics operators (selection, mutation, and crossover) to generate the population child $Q0$ of size N. The elitism is ensured by the comparison between the current population P_t and the preceding population P_{t-1}. The NSGA-II procedure follows (see Algorithm 1).

The NSGA-II algorithm runs in time $O(GN \log^{M-1} N)$, where G is the number of generations, M is the number of objectives, and N is the population size [17]. In addition, our previous experience on multiobjective optimization of soft IP embedded processor [18, 19] emphasizes this choice.

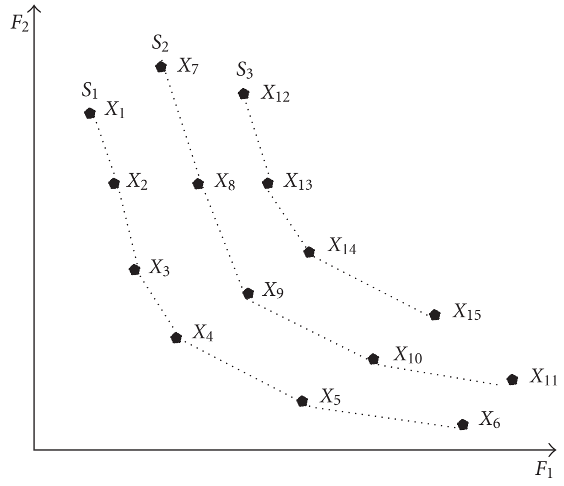


FIGURE 4: Classification of the individuals in several fronts according to the Pareto rank (list of Pareto sets).

```
$R_t = P_t U Q_t$ # combine parent and children
      population
$F$ = fast-nondominated-sort ($R_t$) # $F$ all
      nondominated fronts sets
$P_{t+1} = \emptyset$ and $i = 1$ # initialization
until $|P_{t+1}| + |F_i| \le N$ # till parent pop is filled
   Crowding-distance-assignment ($F_i$) # compute
   distance in $Fi$
   $P_{t+1} = P_{t+1} U F_i$ # include $i$th nondominated
          front in the parent pop
   $i = i + 1$ # check the next front for inclusion
Sort $(F_i, <_n)$ # Sort in descending order using $<_n$
$P_{t+1} = P_{t+1} U F[1 : (N - |P_{t+1}|)]$ # Choose the first
          $(N - |P_{t+1}|)$ elements
$Q_{t+1}$ = make-new-pop ($P_{t+1}$) # apply genetic
          operators to create new pop $Q_{t+1}$
$T = t + 1$ # increment to next generation
```

ALGORITHM 1: NSGA-II.

4.3. MOCDEX

It is clear that MOEAs such as NSGA-II requires the evaluation of individuals (MPSoC configurations) with regard to the 3 objectives considered, BRAM, slices and number of cycles Although, BRAM and slices, could be estimated, we advocate the full use of design automation tools including place and route to access this information. Indeed, for complex systems on large platform FPGA place and route impact cannot be overlooked and can hardly be estimated with sufficient accuracy to be used in an automatic multiobjective design space exploration tool. The execution time of multiprocessor on chip can be obtained through simulation either at RTL level which would be prohibitive for large design space exploration without massive use of computing resources (compute farms) or at TLM level (SystemC) as often advocated [20, 21].

However although SystemC level simulation has been regularly proved to outperform RTL VHDL level simulation, it does not outperform actual execution on FPGA. We argue that for large scale MPSOC, FPGA platform represents an opportunity to both reduce simulation time through actual execution and increase the design space exploration through this reduction of the evaluation of each MPSOC configuration. Our proposal follows.

MOCDEX (general)

(1) Generate random population of MPSOC configurations within soft IP parameters constraints.
(2) For all configurations,
    (a) generate hardware/software platform specification files,
    (b) generate through system EDA and IPs HW/SW model of the MPSOC,
    (c) synthesize/place and route MPSOC configuration using EDA tools,
    (d) record place and route reports,
    (e) download configuration file on FPGA platform,
    (f) execute MPSOC configuration and record execution clock cycles,
    (g) rank the solution.
(3) Generate new population using MOEA algorithm.
(4) Is the Pareto front satisfactory or the number of generations reached if no goto 3?
(5) Final Pareto front MPSOC configurations are available for selection.

As shown in Figure 5, both the DSE and physical design are executed on a host PC while the execution is achieved on a PCI-based FPGA platform which communicates execution results to the host.

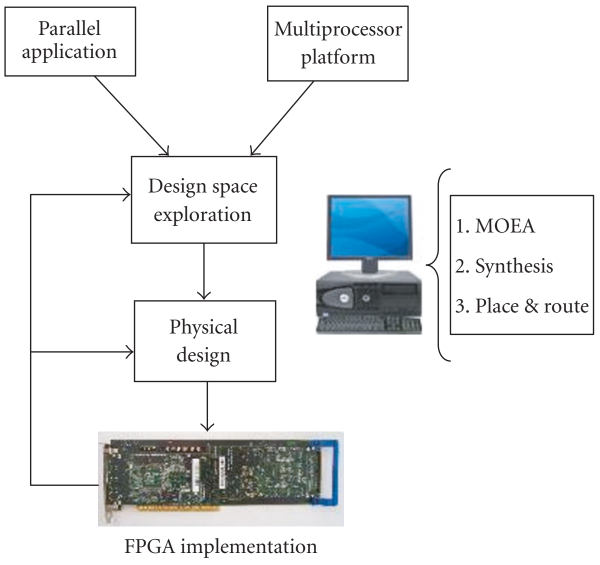


Figure 5: MOCDEX MPSOC exploration flow.

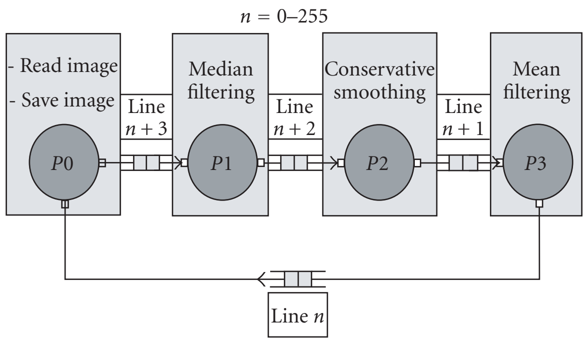


Figure 6: Image filtering application multiprocessor platform distribution.

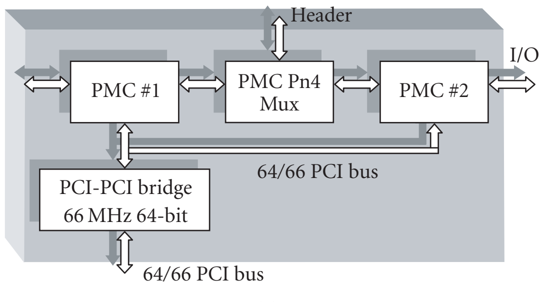
5. CASE STUDY AND VALIDATION

The previously described design flow has been applied in the framework of Xilinx FPGA platforms.

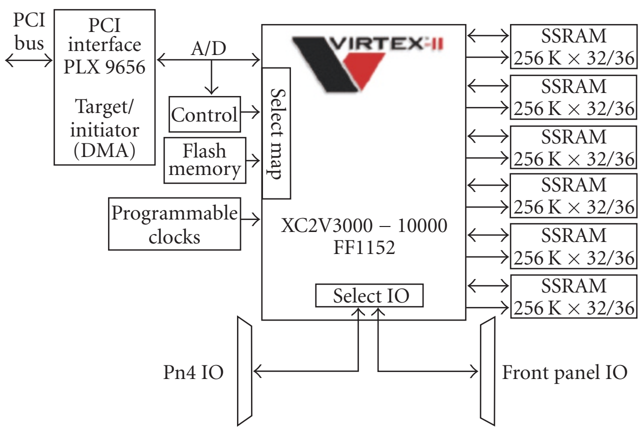
5.1. Image filtering application

A design of four Xilinx MicroBlaze processors, communicating with eight FSL channels in a mesh topology and executing image filtering algorithms, was implemented at 100 MHz. This application was chosen because it requires extensive data processing and data communication among the filters for a good and fast testing of our exploration framework.

Figure 6 shows our filtering methodology. As we can see, the execution is achieved in a pipelined way where image lines are sent from a processor to another as soon as the previous processor has finished its work on it. Obviously, this type of execution makes us save a significant amount of time and memory which are often the major constraints for embedded systems in general and for our platform in particular. Indeed, performing this task in a pipelined way allows us to have a maximum of three image lines stored in the associated processor's memory rather than the whole image. The rest of the image lines will enter the FIFOs (FSLs) of their respective processors one by one. The processor $P0$ in Figure 6 receives image data from the host computer through the PCI bus. Once it receives the data it immediately sends it to the next processor which is $P1$. $P1$ performs a median filtering which results in noise reduction from the image. It is performed on a 3-by-3 pixel window where the center pixel value is replaced by the median of the neighboring pixel values. This value is obtained by sorting the pixels based on their numerical values and then replacing the pixel to be processed by the middle value. The processor $P2$ fetches the line coming from $P1$ and performs a conservative smoothing on it which is an operation that preserves the high spatial frequency details. Finally, the third processor $P3$ performs a mean filtering which consists of very simple method used for noise reduction where the pixel to be processed is replaced by the average



(a)



(b)

Figure 7: Alpha-data ADM-XRC-II and ADC-PMC boards.

Table 2: Multiprocessor on chip design space.

Procs	FSL1Out	FSL2Out	D-Cache	I-Cache
MB0	16...2048	16...2048	512...4096	512...4096
MB1	16...2048	16...2048	512...4096	512...4096
MB2	16...2048	16...2048	512...4096	512...4096
MB3	16...2048	16...2048	512...4096	512...4096

Table 3: Xilinx virtex-II XC2V 8000 resources.

XC2V8000	Values
Slices	46 952
BRAM (18 Kbits)	168
18 × 18 multipliers	168
DCM	12
Max. Dist RAM Kb	1456

value of its neighbors. Due to the different amount of computations required by each filter, it results in different workload for each processor. Thus the execution time for each algorithm differs and hence involves an unequal FIFOs occupancy. Therefore, the application used has to be naturally unbalanced to thoroughly analyze the problem. The problem at hand is to optimally distribute the limited on chip embedded memory among the embedded processors cache memories (instruction, data) and the communication FIFOs while optimizing execution time and area. The design space for this problem is specified in Table 2.

The possible number of different configurations is given by the product of the number of distinct configurations for each configurable architectural parameter. Each cache memory may have up to 4 different sizes and each FIFO up to 8 different sizes. The total design space represents $(4 \times 4 \times 8 \times 8)^4 = 2^{40}$ configurations. If each configuration evaluation would require 1 second, the total evaluation time would be 34 865 years of evaluation. Clearly an exhaustive evaluation technique is unfeasible and multiobjective optimization techniques are able to efficiently prune this design space while simulation is clearly outperformed by direct execution on large scale FPGA devices.

5.2. *Alpha-data environment*

For the implementation of MOCDEX we used the alpha-data hardware and software environment.

5.2.1. *Alpha data hardware environment*

The alpha-data hardware environment described in Figure 7 is composed by (1) the ADC-PMC and (2) the ADM-XRC-II. The ADC-PMC is a dual PMC adapter for PCI. It supports 64-bit 66 MHz primary and secondary PCI via an Intel 21154 PCI-PCI bridge device. The ADM-XRC-II is a high performance reconfigurable PMC (PCI mezzanine card) based on the Xilinx Virtex-II range of platform FPGAs. Features include high-speed PCI interface, external memory, high-density I/O, programmable clocks, temperature monitoring, battery backed encryption, and flash boot facilities.

On board clock generator provides a synchronous local bus clock for the PCI interface and the Xilinx Virtex-II FPGA. A second clock is provided to the Xilinx Virtex-II FPGA for user applications and can be free running or stepped under software control. Both clocks are programmable and can be used by the Virtex clock. The user clock has a maximum value of 100 MHz. The ADM-XRC-II uses a Xilinx XC2V8000-6 FF1152 device [22] whose characteristics are described Table 3.

5.2.2. *Alpha-data software environment*

The ADM-XRC SDK is a set of resources including an application-programing interface (API) intended to assist the user in creating an application using one of Alpha-data's ADM-XRC range of reconfigurable coprocessors. The API

Table 4: ADM XRC SDK API functions.

Group	Application
Initialization	ADMXRC2_CloseCard
	ADMXRC2_OpenCard
	ADMXRC2_OpenCardByIndex
	ADMXRC2_SetSpaceConfig
FPGA configuration through PCI	ADMXRC2_ConfigureFromBuffer
	ADMXRC2_ConfigureFromBufferDMA
	ADMXRC2_ConfigureFromFile
	ADMXRC2_ConfigureFromFileDMA
	ADMXRC2_LoadBitstream
	ADMXRC2_UnloadBitstream
Data transfer PC = FPGA board	ADMXRC2_BuildDMAModeWord
	ADMXRC2_DoDMA
	ADMXRC2_DoDMAImmediate
	ADMXRC2_MapDirectMaster
	ADMXRC2_Read
	ADMXRC2_ReadConfig
	ADMXRC2_SetupDMA
	ADMXRC2_SyncDirectMaster
	ADMXRC2_UnsetupDMA
	ADMXRC2_Write
	ADMXRC2_WriteConfig
Interrupt handling	ADMXRC2_RegisterInterruptEvent
	ADMXRC2_UnregisterInterruptEvent

makes use of a device driver that is normally not directly accessed by the user's application. The API library described in Table 4 takes care of open, close, and device I/O control calls to the driver. The ADM-XRC SDK is designed to be thread-safe. Table 4 describes the main API functions which allow initializing the board, configuring the FPGA though the PCI bus, and transfering data between the FPGA and the host computer and the interrupt handling.

Clearly since MOCDEX explore the design space by implementing on FPGA new multiprocessor configurations the FPGA is reconfigured through the PCI bus from the main program by executing the ADM-XRC SDK FPGA reconfiguration API using the bitfile generated from EDK synthesis and place and route. Resulting execution number of cycles are provided as well through the PCI bus to the host using ADM-XRC SDK data transfer API.

5.3. *Xilinx EDK tools*

The embedded development kit (EDK) bundle is an integrated software solution for designing embedded processing systems.

Table 5 and Figure 8 describe the use of each configuration file in the process of hardware platform generation, software platform generation, and software application and creation.

The MHS file defines the system architecture, peripherals, and embedded processors. It also defines the connectivity of the system, the address map of each peripheral in the system, and configurable options for each peripheral. The MHS file can be defined through XPS Gui wizards. However for the time being Xilinx wizards do not allow the design of multiprocessors platforms and therefore they should be defined directly in the MHS file. It is clear that in the purpose of design space exploration of multiprocessor architecture the MHS file is the prime target of modifications. Changing parameters value in the MHS file generates a new multiprocessor configuration and invoking the XPS tool in no window mode from a main program allows the generation of the multiprocessor netlist. Table 6 provides examples of MHS file parts.

5.4. *Exploration flow description*

The proposed automatic design flow described in Figure 5 can be applied in the framework of Xilinx EDA tools and the Alpha-data environment. The flow is mainly composed of 3 parts: (1) architecture design space exploration engine (DSE), (2) physical design, and (3) FPGA platform PCI board. The architecture design space exploration part controls the whole flow and runs on a host PC. First based on the user specified design space parameters and parameters range, the DSE specifies the architectural parameters of the multiprocessors configurations to be evaluated then translates those parameters into platform EDA design tool input file specifications. In our case,

(1) *MOCDEX for Xilinx FPGA platform*,
(2) generate random population of MPSoC configurations (caches and FSL variations),
(3) for all configurations,
    (a) generate hardware/software platform specification files (mhs, mpd, pao, mss, mld, mdd, files),
    (b) generate through Xilinx system XPS and Xilinx IPs HW/SW model of the MPSOC,
    (c) synthesize/place and route MPSOC configuration using Xilinx ISE 6.3,
    (d) record place and route reports generated from Xilinx ISE 6.3,
    (e) download configuration file on FPGA Alpha-data platform using ADM-XRC SDK API,
    (f) execute MPSOC configuration and record execution clock cycles using ADM-XRC SDK API,
    (g) rank the solution,
(4) generate new population using NSGA-II algorithm,
(5) is the Pareto front satisfactory or the number of generations reached if no goto 3?
(6) final Pareto front MPSOC configurations available for selection.

The Xilinx system EDA tools Xilinx platform studio (XPS) is ran in no window mode with all batch commands launched from a *C* main program. Those input file specifications are used to control the physical design part of the implementation by synthesizing, placing, and routing the multiprocessor configurations onto FPGA platform devices. The generated

Table 5: EDK specifications files.

Files	Description	Comments
MHS	Microprocessor hardware specification	The MHS defines the hardware component
MSS	Microprocessor software specification	The MSS contains directives for customizing libraries, drivers, and file systems
MDD	Microprocessor driver definition	An MDD file contains directives for customizing software drivers
MPD	Microprocessor peripheral definition	The MPD defines the interface of the peripheral
MLD	Microprocessor library definition	the MLD contains directives for customizing software libraries and operating systems
PAO	Peripheral analyze order	Contains a list of HDL files that are needed for synthesis, and defines the analyze order for compilation.

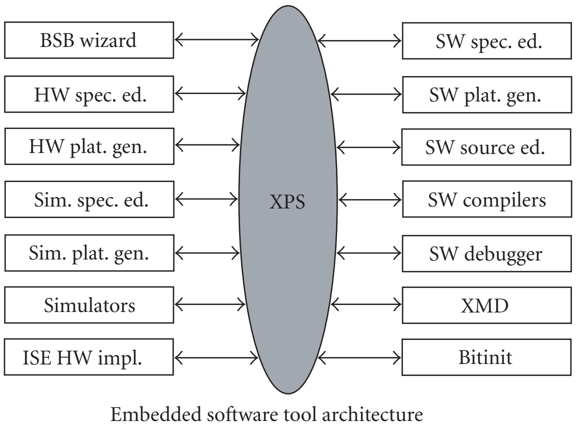


Figure 8: Xilinx EDK (XPS Xilinx platform studio).

FPGA configuration bitstream is downloaded on the FPGA device for execution and performance evaluation of the multiprocessor. The board hosting the FPGA device is an Alphadata PCI FPGA board [3]. The implementation area and resources of the multiprocessor configurations are provided by the design automation tools composing part (2) while performance results in number of clock cycles are obtained from the actual execution of the multiprocessor configurations. These informations are automatically fed back to the DSE engine which runs on the host through the PCI bus.

The number of cycles are obtained directly from the execution, thanks to a timer connected to the MicroBlaze (MB0) OPB bus, which counts the number of clock cycles. After that, the execution time results are communicated to the host PC using an IP which bridges the MicroBlaze OPB bus to the PCI host bus. These results (occupied slices, occupied BRAM, and the execution time) are then injected as feedback input to the evolutionary algorithm for the next generation run. For this work we initially executed two explorations where the first consisted of a population size of 22 individuals and 10 generations (242 implementations with the initialization generation).

6. EXPLORATION RESULTS

6.1. Flow execution results

Figures 10 and 11 describe the corresponding results of these implementations. Figure 10(b) represents Pareto solutions for the second exploration where we attempted to increase the population size to 30 individuals and the number of generations to 14 in order to observe the behavior of the evolutionary algorithm for bigger explorations. From the results of second exploration it is obvious that the algorithm is converging to optimal solutions showing that for larger population size and generation size, potential of convergence is increased in NSGA-II algorithm as was expected. From the two preceding exploration flow executions, it appears as expected since we focused on embedded memories that the number of occupied slices does not vary much across multiprocessor configurations. However the variations are much more significant concerning both the number of occupied BRAMs and the execution time. So we decided to continue the execution of the proposed exploration flow in order to see its evolution.

Table 6: MHS file parts: Microprocessor IP, FSL IP, BRAM controller IP.

MicroBlaze processor	FSL communication	BRAM controller
BEGIN MicroBlaze	BEGIN fsl_v20	BEGIN lmb_bram_if_cntlr
PARAMETER INSTANCE = MicroBlaze_0	PARAMETER INSTANCE = fsl_v20_7	PARAMETER INSTANCE = ilmb_cntlr3
PARAMETER HW_VER = 3.00.a	PARAMETER C_FSL_DEPTH = 8	PARAMETER HW_VER = 1.00.b
PARAMETER C_FSL_LINKS = 2	PARAMETER HW_VER = 2.00.a	PARAMETER C_BASEADDR
BUS_INTERFACE MFSL0 = fsl_v20_2	PARAMETER C_EXT_RESET_HIGH = 0	= 0 × 00000000
BUS_INTERFACE SFSL0 = fsl_v20_1	PARAMETER C_IMPL_STYLE = 1	PARAMETER C_HIGHADDR
BUS_INTERFACE DLMB = dlmb0	PARAMETER C_USE_CONTROL = 0	= 0 × 00003fff
BUS_INTERFACE ILMB = ilmb0	PORT SYS_Rst = lreseto_l	BUS_INTERFACE SLMB = ilmb3
BUS_INTERFACE DOPB = mb_opb0	PORT FSL_Clk = lclk	BUS_INTERFACE BRAM_PORT
BUS_INTERFACE IOPB = mb_opb0	PORT FSL_M_Clk = lclk	= ilmb_port3
PORT INTERRUPT = Interrupt 0	PORT FSL_S_Clk = lclk	END
PORT CLK = lclk	END	
END		

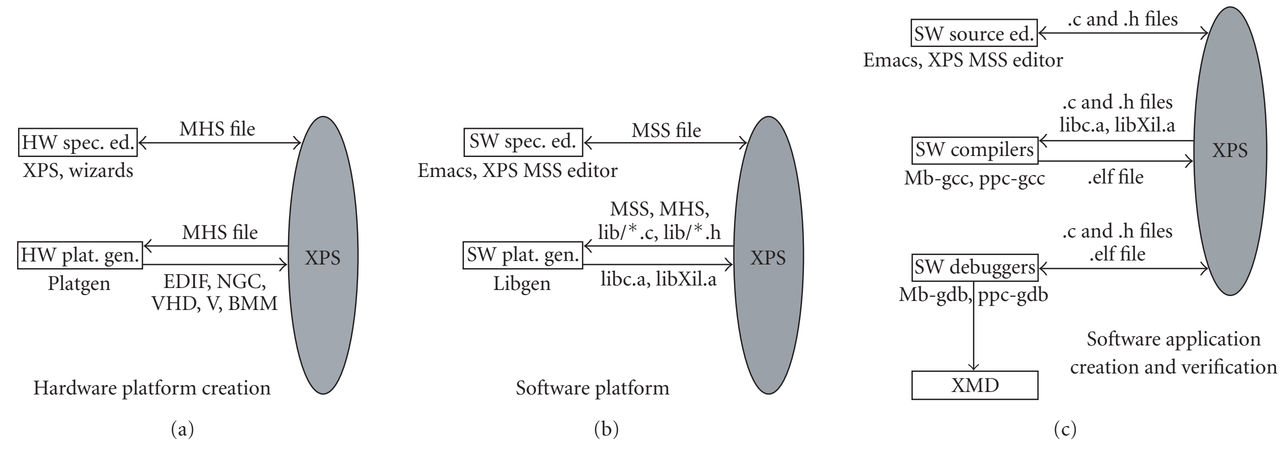


Figure 9: Xilinx EDK. (a) Hardware platform generation. (b) Software platform. (c) Simulation and verification.

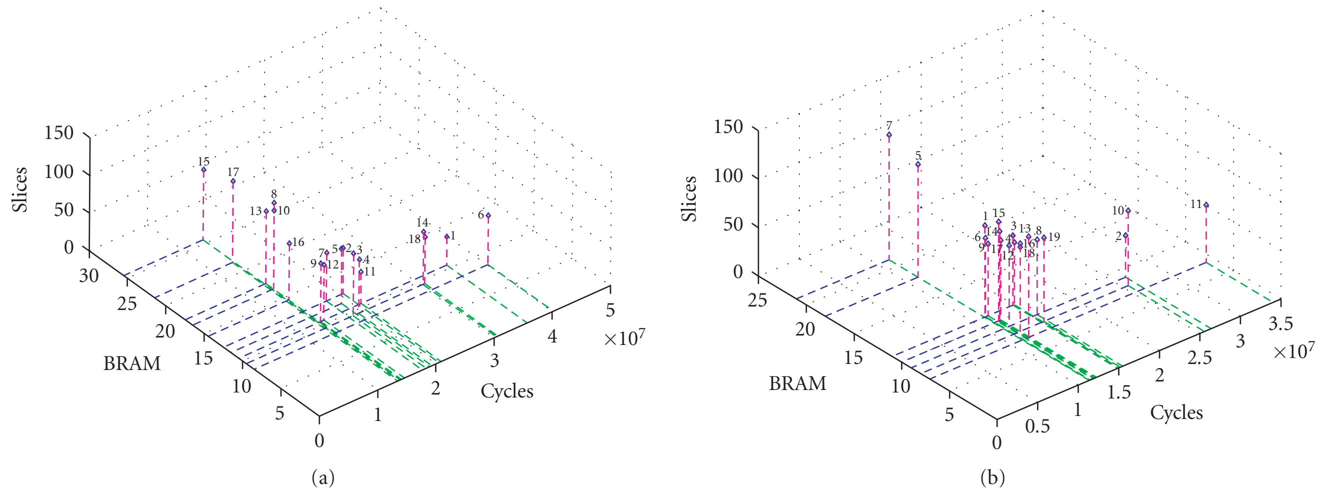


Figure 10: (a) For 10 generations-popsize = 22. (b) For 14 generations-popsize = 30.

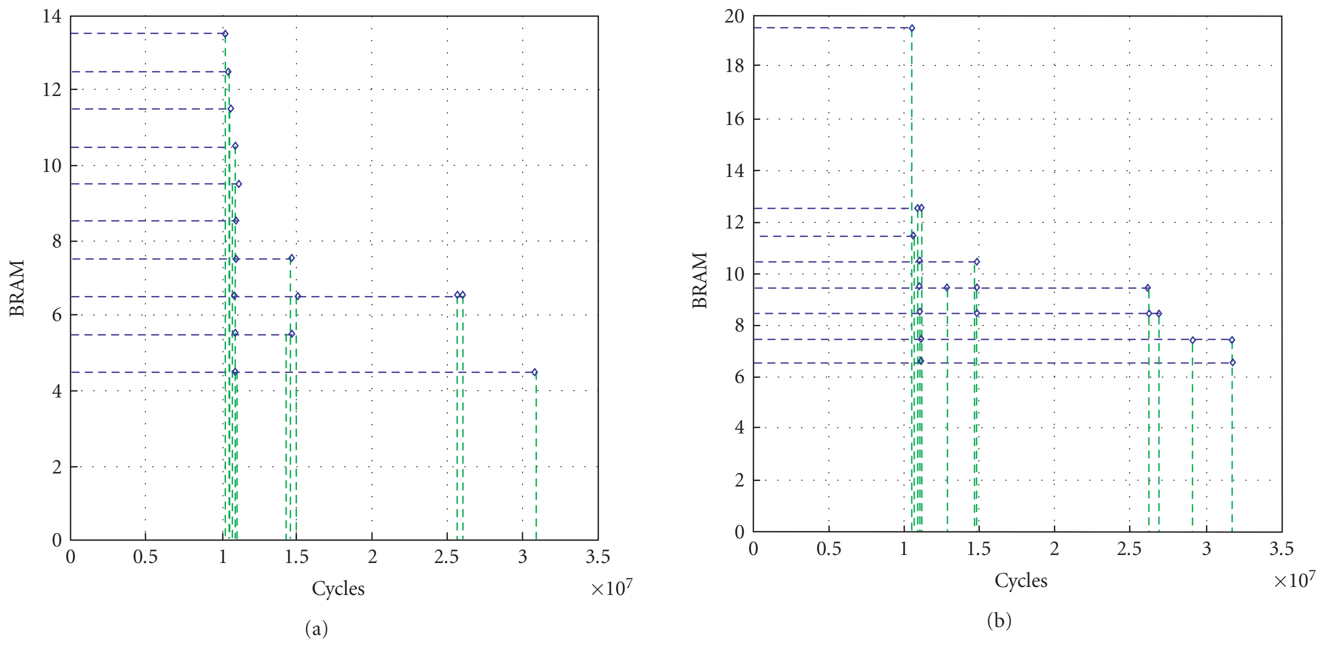


FIGURE 11: (a) For 30 generations-popsize = 30. (b) For 60 generations-popsize = 30.

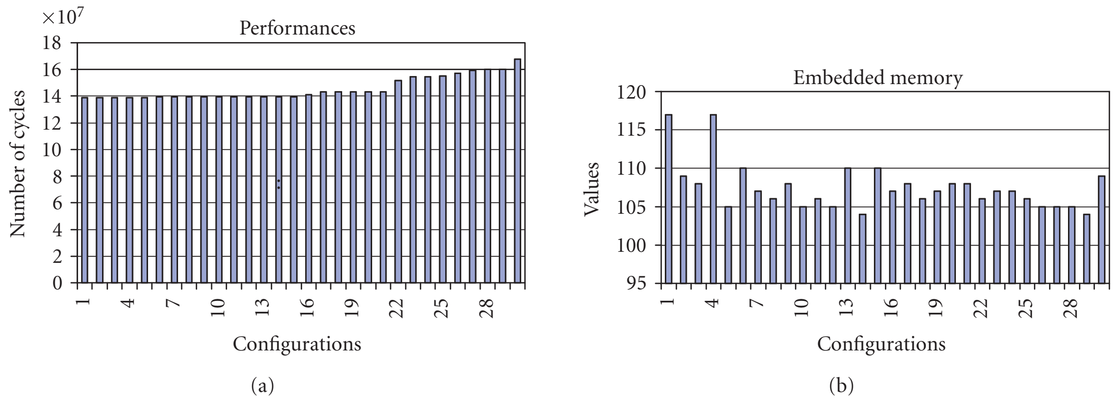


FIGURE 12: Pareto front. (a) Pareto front performance distribution. (b) Pareto front BRAM distribution.

For this second part of the exploration, we fixed the population size to 30 individuals and changed the number of generation to 30 and finally 60 generations. The results for each execution are, respectively, described in Figures 11(a) and 11(b). From these different figures we can clearly observe that the NSGA-II evolutionary algorithm tends to converge to the optimal Pareto solutions front which proves the correct implementation of the algorithm. The figures show different execution times for the same BRAM occupation meaning that using more BRAM will not systematically result in performance improvements.

However, to achieve better results BRAM resources need to be well distributed among the IPs where it would be used for getting optimal resource utilization.

Figure 12 shows the distribution of performance in the final Pareto front and clearly few configurations demonstrate superior performance while BRAM distribution for the same front demonstrates an uneven use of BRAM. This clearly shows the impact of BRAM careful distribution.

Examples of final Pareto front configurations are given in Table 7. The configurations chosen represent, respectively, 69.64%, 61.90%, and 64.88% of all BRAM resources. 11.11% BRAM reduction is obtained in the second configuration for a 0.004% increase in execution time while a 6.8% BRAM reduction is obtained in the third configuration for a 0.009% increase in the execution time.

6.2. *Flow execution time*

The results achieved in the previous section required the performance evaluation of 3120 different multiprocessor

Table 7: The design space associated with those parameters (74 × 118, thus 514 675 673 281 different configurations) requires 16 321 years of simulation for 1 minute simulation per configuration.

(a) Cycles: 138 974 816 BRAM: 109.

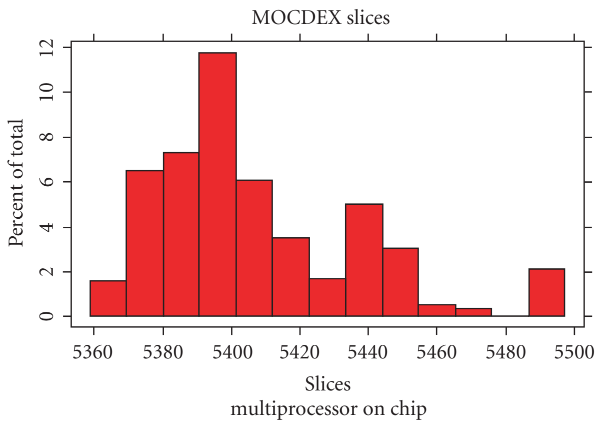
Procs	FSL1Out	FSL2Out	D-Cache	I-Cache
MB0	2048	2048	1024	4096
MB1	512	512	1024	1024
MB2	2048	512	2048	2048
MB3	1024	1024	4096	4096

(b) Cycles: 138 844 064 BRAM: 117.

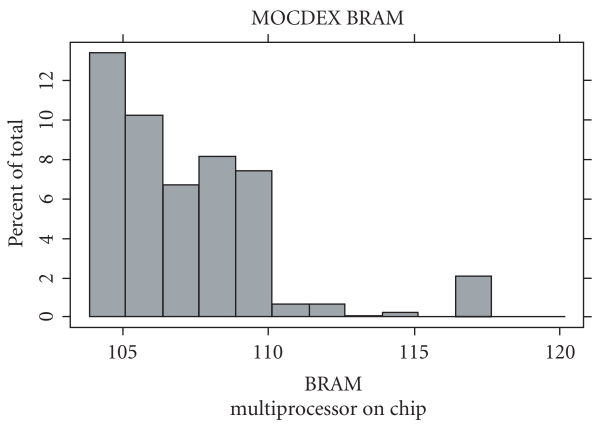
Procs	FSL1Out	FSL2Out	D-Cache	I-Cache
MB0	2048	128	2048	2048
MB1	256	32	2048	512
MB2	512	16	4096	512
MB3	1024	32	512	2048

Table 8: Flow execution time direct execution versus simulation.

Flow main steps	Functions	Time
Multi-objective evolutionary algorithm (ms)	Indi. Gene.	190
	Obj functions eval.	293
	Selection	0.116
	Crossover	0.033
	Mutation	1.118
Synthesis (sec)	Synthesis	523.503
	P and *R*	655.174
	P/*R* & Bitgen	797.856
Evaluation	Exploration 60 × 30	
	Sim. 64 × 64	2250 days
	Direct exec. 256 × 256	1.39 hour



(a)



(b)

Figure 13: Explored design space. (a) Slices histogram. (b) BRAM histogram.

on-chip configurations. These evaluations have been cycle-accurate after actual implementation on single-chip large scale FPGA devices. Contrary to traditional board-based multiprocessor, multiprocessors on chip are implemented on single chip; and due to the complexity of these architectures and the scale of the target devices, it is not possible to overlook the impact of place and route on the number of cycles required for various operations and on the cycle time.

It results from this fact that comparing different multiprocessors on chip configurations on the number of execution cycles is meaningless if one does not take into account the impact of place and route on each distinct configuration resulting from actual implementation. From this point mainly two alternatives exist: (1) post place and route simulation which will accurately represent the multiprocessor on chip behavior, and (2) emulation through direct execution. We conducted cycle accurate simulations using a powerful multi-language (SystemC, VHDL, Verilog-HDL) simulator ModelSim 6.0. Indeed, ModelSim 6.0 can handle large and complex designs and allow their simulation in a post-synthesis and post-place and route modes. Table 8 describes the very important time savings while using direct execution instead of simulation. Simulation would require 2250 days of simulation versus 1.39 hour for direct execution.

In order to reach the same evaluation speed at this level of accuracy it would require a compute farm (grid computing) of well over 25 000 workstations. The proposed flow execution time is obviously very competitive with regard to SystemC approaches [20, 21] for platform-based design. Similar observations have been drawn for embedded processors design space exploration [18, 19].

7. EXPLORED DESIGN SPACE STATISTICAL ANALYSIS

If we analyze in detail the complexity landscape of such a design space exploration we obtain the configurations distribution found in Figures 13 and 14. Clearly from these histograms we see that slices, BRAM, and performance (execution time) distributions in the explored design space are very different and demonstrate that the design space exploration was not confined in a limited subspace but explored a large diversity of multiprocessor configurations. The explored design landscape is given in Figure 15.

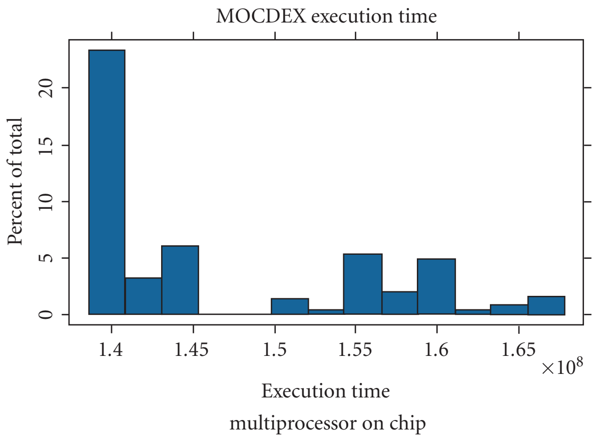


FIGURE 14: Explored design space execution time histogram.

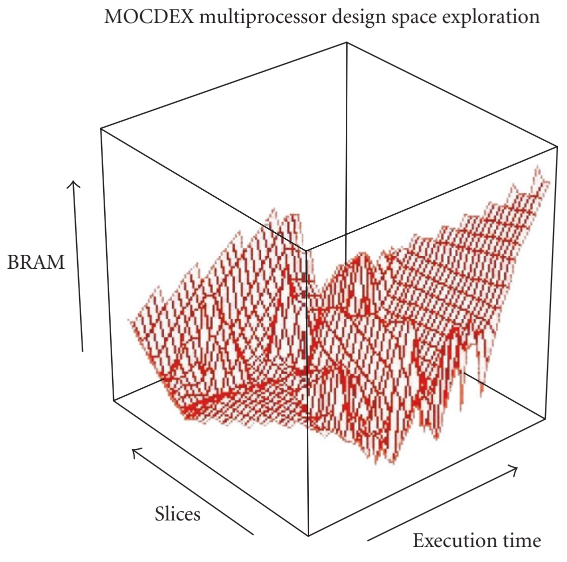


FIGURE 15: MOCDEX explored design space.

Figure 15 demonstrates the complexity of the design landscape and emphasizes the need to match this complexity with appropriate applied mathematics optimization techniques.

8. CONCLUSION

The design complexity of multiprocessors on chip requires efficient design methodologies. We propose in this paper a novel technique which fully integrates architectural design space exploration with design automation tools, where all area and performance results are obtained from actual post-synthesis place and route and actual execution on large scale FPGA platforms. To the best of our knowledge, our work is the first to fully integrate and therefore close the gap between design automation tools and architecture design space exploration technique in a multiobjective constraints paradigm with actual execution for all multiprocessor on chip configurations explored during the design space exploration process. It is important to note that actual execution reduces exploration time and can be exploited for either reducing design cycle time (i.e., TTM) and/or exploring even larger design space by including additional parameters. This work can be easily extended to include more parameters at various abstraction levels from architecture to circuit allowing interesting tradeoffs between usually uncorrelated various abstraction levels in the general design flow.

REFERENCES

[1] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*, Springer, New York, NY, USA, 2002.

[2] C. A. C. Coello, D. V. Veldhuizen, and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, vol. 5 of *Genetic Algorithms and Evolutionary Computation*, Kluwer Academic, Dordrecht, The Netherlands, 2002.

[3] Alpha-Data, ADM-XRC-II PCI mezzanine card, http://www.alpha-data.com.

[4] D. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann, San Francisco, Calif, USA, 1999.

[5] A. A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chips*, Morgan Kaufman, San Francisco, Calif, USA, 2004.

[6] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya, "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 518–523, Las Vegas, Nev, USA, June 2001.

[7] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha, "Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors," in *Proceedings of the 18th IEEE International Conference on VLSI Design*, pp. 551–556, Kolkata, India, January 2005.

[8] Y. Jin, N. Satish, K. Ravindran, and K. Keutzer, "An automated exploration framework for FPGA-based soft multiprocessor systems," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES '05)*, pp. 273–278, New York, NY, USA, September 2005.

[9] N. K. Bambha and S. S. Bhattacharyya, "Joint application mapping/interconnect synthesis techniques for embedded chip-scale multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 99–112, 2005.

[10] Xilinx, Embedded system tools guide, http://www.xilinx.com/ise/embedded/edk_docs.htm.

[11] Xilinx microblaze soft core processor, http://www.xilinx.com/ise/embedded/mb_ref guide.

[12] I. Aouadi, R. B. Mouhoub, and O. Hammami, "System on a programmable chip oriented JPEG-2000 entropy coder implementation for multimedia embedded systems," in *Proceedings of the IEEE International Conference on Consumer Electronics (ICCE '05)*, pp. 447–448, Las Vegas, Nev, USA, January 2005.

[13] R. B. Mouhoub, I. Aouadi, and O. Hammami, "System on programmable chip platform based design of JPEG- 2000 entropy coder," in *Proceedings of the 12th Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI '04)*, pp. 103–106, Kanazawa, Japan, October 2004.

[14] Xilinx Fast Simplex Link IP, http://www.xilinx.com.

[15] C. A. C. Coello, "An updated survey of GA-based multiobjective optimization techniques," *ACM Computing Surveys*, vol. 32, no. 2, pp. 109–143, 2000.

[16] 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. 182–197, 2002.

[17] M. T. Jensen, "Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, 2003.

[18] K. Ghali and O. Hammami, "Embedded processor characteristics specification through multiobjective evolutionary algorithms," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '03)*, vol. 2, pp. 907–912, Rio de Janeiro, Brazil, June 2003.

[19] K. Ghali and O. Hammami, "Embedded processors optimization with hardware in the loop," in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE '04)*, vol. 1, pp. 561–564, Ajaccio, France, May 2004.

[20] F. Fummi, S. Martini, G. Perbellini, and M. Poncino, "Native ISS-SystemC integration for the co-simulation of multiprocessor SoC," in *Proceedings of the IEEE Conference and Exhibition on Design, Automation and Test in Europe (DATE '04)*, vol. 1, pp. 564–569, Paris, France, February 2004.

[21] F. Ghenassia, *Transaction-Level Modeling with SystemC TLM Concepts and Applications for Embedded Systems*, Springer, New York, NY, USA, 2005.

[22] Xilinx Virtex-II Platform FPGA, http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex_ii_platform_fpgas/index.htm

Riad Ben Mouhoub received the Electronics Engineering degree in 2002 from the University of Algiers USTHB. He also received a Master degree in electronic systems and data processing from the University of Paris Sud Orsay in 2003. He currently holds a Doctorate position in electrical and computer engineering from the University of Paris Sud in the Department of Electronics and Computer Engineering at École Nationale Supérieure de Techniques Avancées in Paris (ENSTA). His research interests include design automation, design methodologies for multiprocessor system on programmable chips (MPSoPC), and NoC synthesis. He is a Student Member of the IEEE.

Omar Hammami is an Associate Professor at ENSTA/DGA since 2000. Prior to that he was Assistant Professor from 1991 to 1993 with ENSEEIHT, Toulouse, and Associate Professor with the University of Aizu, Japan, from 1993 to 2000. He received his Ph.D. degree in computer science and electrical engineering from Paul Sabatier University, Toulouse, in 1993 and has since worked in the field of circuits, system level design methodologies, embedded parallel architectures, and system on chip (SOC) for multimedia and wireless communications. He has been involved in numerous international and national research and industrial projects in those areas and have been funded by various government and funding agencies. He is a regular reviewer for various journals (IEEE, EURASIP, etc.) and conferences as Program Committee Member.