International Journal of Reconfigurable Computing

Current Trends on Reconfigurable Computing

Guest Editors: Jürgen Becker, Michael Hübner, Roger Woods, Philip Leong, Robert Esser, and Lionel Torres

Current Trends on Reconfigurable Computing

International Journal of Reconfigurable Computing

Current Trends on Reconfigurable Computing

Guest Editors: Jürgen Becker, Michael Hübner, Roger Wood, Philip Leong, Rob Esser, and Lionel Torres

Contents

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 918525, 1 page
doi:10.1155/2008/918525

Editorial

Current Trends on Reconfigurable Computing

Jürgen Becker,[1] Michael Hübner,[1] Roger Woods,[2] Philip Leong,[3] Robert Esser,[4] and Lionel Torres[5]

[1] *Universität Karlsruhe (TH), 76131 Karlsruhe, Germany*
[2] *Queens University Belfast, Belfast BT7 1NN, Northern Ireland*
[3] *Chinese University Hong Kong, Hong Kong*
[4] *Xilinx Inc., San Jose, CA 95124, USA*
[5] *Le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), Université Montpellier II/CNRS, 161 Rue Ada, 34392 Montpellier, France*

Correspondence should be addressed to Jürgen Becker, becker@itiv.uni-karlsruhe.de

Received 15 December 2008; Accepted 15 December 2008

This special issue covers actual and future trends on reconfigurable computing given by academic and industrial specialists from all over the world. In the issue, Boukhechem and Bourennane address transaction-level modeling, a promising technique for increasingly complex embedded system. They present a technique for modeling, validating, and verifying an open embedded platform which allows faster and more efficient architecture exploration. Puschini et al. argue that future systems will have distributed decision making capability, one of which will be to control the voltage scaling in the device. They apply game theory to the run-time optimization of the frequency of multiprocessor SoC platforms, resulting in a temperature reduction. The work by Amagasaki et al. outlines an approach for varying the granularity of an FPGA logic cell when implementing arithmetic and random logic. The resulting cell gives improved logic depth and needs less reconfiguration data when compared to a Xilinx Virtex-4 device. Siozios et al. have presented tools for exploration of alternative interconnection schemes for 3D FPGAs which provide partitioning, placement and routing, and power estimation. The work on medical image registration by Dandekar et al. outlines a novel multiobjective optimization strategy for exploring the tradeoff between FPGA resources and implementation accuracy. Craven and Athanas present work on a high-level development environment for reconfigurable designs, that leverages an existing high-level synthesis tool to enable the design, simulation, and implementation of dynamically reconfigurable hardware based on a C specification. Guillemenet et al. looked at the integration of field-induced magnetic switching and thermally assisted switching magnetic RAMs in FPGA design, highlighting reductions in both power consumption and configuration time at power up when compared to classical SRAM-based FPGAs. The work by Steiner and Athanas describes the computing infrastructure that needs to be included in order to allow a system to change its operation without requiring outside intervention. Together, we hope that this special issue will serve as an introduction to users who have newly joined the community as well as provide specialists with recent results in this field of research.

Jürgen Becker
Michael Hübner
Roger Woods
Philip Leong
Robert Esser
Lionel Torres

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 902653, 10 pages
doi:10.1155/2008/902653

Research Article

SystemC Transaction-Level Modeling of an MPSoC Platform Based on an Open Source ISS by Using Interprocess Communication

Sami Boukhechem and El-Bay Bourennane

UMR CNRS 5158, University of Burgundy, 9 Avenue Alain Savary B.P: 47870, 21078 Dijon Cedex, France

Correspondence should be addressed to Sami Boukhechem, sami.boukhechem@u-bourgogne.fr

Received 29 February 2008; Revised 20 May 2008; Accepted 18 August 2008

Recommended by Michael Hubner

Transaction-level modeling (TLM) is a promising technique to deal with the increasing complexity of modern embedded systems. This model allows a system designer to model a complete application, composed of hardware and software parts, at several levels of abstraction. For this purpose, we use systemC, which is proposed as a standardized modeling language. This paper presents a transaction-level modeling cosimulation methodology for modeling, validating, and verifying our embedded open architecture platform. The proposed platform is an open source multiprocessor system-on-chip (MPSoC) platform, integrated under the synthesis tool for adaptive and reconfigurable system-on-chip (STARSoC) environment. It relies on the integration between an open source instruction set simulators (ISSs), OR1Ksim platform, and the systemC simulation environment which contains other components (wishbone bus, memories,..., etc.). The aim of this work is to provide designers with the possibility of faster and efficient architecture exploration at a higher level of abstractions, starting from an algorithmic description to implementation details.

1. Introduction

It has recently become possible to create complex embedded systems called the multiprocessor system on chip (MPSoC), usually used for embedded applications. These systems contain several microprocessors, memories (shared, private), shared busses, and peripherals integrated in a single die [1]. As a consequence, the system designer is confronted with new challenges and difficulties related to the integration of such complex systems. So before any implementation, it is necessary to validate by simulation the system to be implemented. The chosen TLM simulation framework permits rapid exploration of several solutions containing different descriptions of the system components.

For this purpose, a hardware/software TLM cosimulation is used to validate the behavior for both the hardware and the software components of embedded systems, as well as the interaction between them. Moreover, TLM cosimulation also permits the performance evaluation of the whole system at the earlier stages of the design flow before building a prototype, which is faster than HDL register-transfer level (RTL) simulation [2, 3].

Traditionally, mixed language cosimulators are used [4] for simulation which generates a communication overhead between different simulators, often resulting in a significant degradation in the execution time [5]. It is thus necessary to use the same language for modeling software and hardware, and simulating these models at system level in a unified systems design approach. Many research and commercial products provide cosimulation environments which combine ISS, HDL simulators, or systemC models [6, 7].

This motivated our choice of systemC as the modeling and simulation environment for our MPSoC platform. SystemC has become a standard in system level design; it is one of the leading C/C++ design environments, and is an open source, free simulation environment [8].

In addition, TLM cosimulation becomes easier and more efficient, because the entire system can be simulated

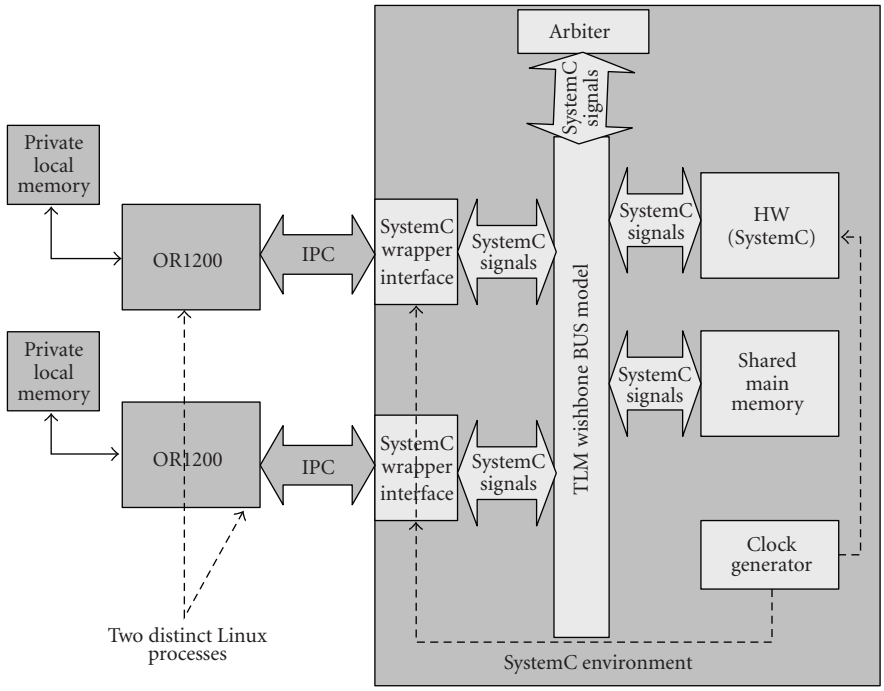


Figure 1: STARSoC platform.

within a single simulation engine (systemC). It permits the simulation at different abstraction levels starting at a very high level of functional description and continuing after refining over time, to synthesizable Register-Transfer Level style, and even combines different levels in one model. The systemC simulation kernel also treats parallel execution and provides functions required to model hardware timing and concurrency [8].

The ISSs in our platform run as distinct UNIX processes on the host system. They communicate between themselves and with hardware components through their systemC interface wrappers which communicate with the other platform components via abstract systemC communication channels [9, 10]. The instruction set simulators based on C language communicate with their systemC interface wrappers via interprocess communication (IPC) [11] primitives.

In this context, there are many related works, such as [12]. The major contribution of this paper is to develop the cosimulation environment between OR1Ksim [13] and systemC. We used IPC (PIPE) in order to provide MPSoC models for an OpenRISC processor at a higher level of abstraction. These can be used in order to accelerate the validation, performance analysis, and architecture exploration for our project, called STARSoC. It is particularly used for evaluating the hardware-software partitioning and also for comparing the system modeled at a high level of abstraction (TLM) with the register-transfer level during software prototyping.

Our objective in this paper consists in comparing the three abstraction levels which are traditional register-transfer level modeling and transaction-level modeling at instruction accurate level and cycle-accurate level.

In our case study, we have used an open source ISS derived from the OR1Ksim simulation platform which is designed for monoprocessor simulation. For the use in the case of the simulation of multiprocessor systems, we connect two ISSs with systemC communication platform models, by using interprocess communication. Thus, it is very easy to add or to remove a processor from the MPSoC design. The interconnection models and other hardware components can be modeled in systemC or any other hardware description language. The interconnection is based on a standard Wishbone bus [13]. Our communication model uses the shared memory communication mechanism. All processors are simulated at the same abstraction level, for each TLM level [14]. The rest of the paper is organized as follows. We begin in Section 2 with a brief definition of STARSoC design flow and transaction-level modeling. In Section 3, we discuss the simulation platform used in this work. Section 4 describes the bus architecture. Section 5 is devoted to the study of our transaction-level modeling steps of the wishbone bus. Section 6 introduces the communication model adopted in our work. In Section 7, we provide some explanations for the modified instruction set simulator based on OR1Ksim. Section 8 presents some results of experiments we obtained by simulation. We finish with a conclusion in Section 9.

2. STARSoC Design Flow Overview

An overview of the STARSoC design flow [15] is shown in Figure 2. The input description consists of a set of communicating parallel software and hardware processes described in C-code [16]. We specify the number of reconfigurable processors (OpenRISC1200) and the list of peripheral

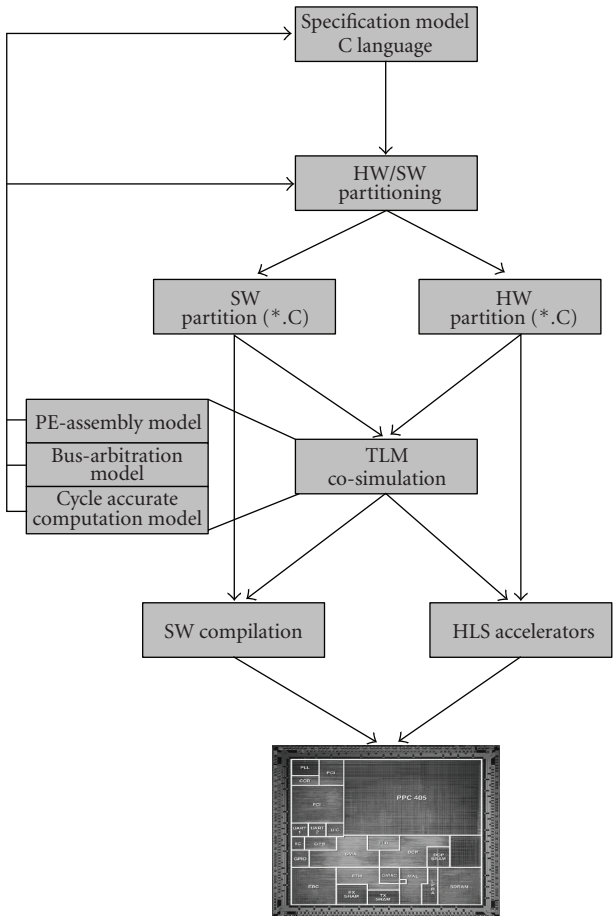


Figure 2: STARSoC design flow.

input/output components connected to each processor. After hardware-software partitioning, the hardware part is synthesized into register-transfer level architecture, and the software part is distributed to the available processors.

The hardware and software partitions are defined by the user. The software part will be reinstrumented to make appropriate calls to the hardware component synthesized via the communication system bus. The bus-based communication architecture will be synthesized to interconnect the software processor(s) and the hardware coprocessor(s). We use the GCC compiler to synthesize the machine code of the software processes and download it into the program memory of each processor in the generated MPSoC platform.

In this work, we focus on TLM abstraction as defined in the following.

2.1. Transaction-Level Modeling

Currently, transaction-level modeling is widely referred to in system-level design literature. It permits a fast simulation and performance evaluation of a complex System on Chips (SoCs) earlier in the design flow, with a more modular and efficient code. This reduces the time-to-market compared with RTL and ensures practical gains for design, because TLM is less detailed. Timing details can be incorporated into these models to allow performance estimation and architecture exploration before the RTL (HDL/systemC) code is generated.

We have several definitions concerning the exact place of TLM in the simulation level. TLM is not a single abstraction level but involves several abstraction levels (multilevel model). We can refine the models over time to include more information. In most cases, TLM is defined above the RTL [17, 18]. Gai and Gajski [14] clearly define four transaction level abstraction models, where the communication and the computation are explicitly separated. The system is represented as a set of communicating processes. These processes perform computations and communicate with other processes through an abstract channel. The different transaction levels defined by [14] are PE-assembly model, bus-arbitration model, time-accurate communication model, and cycle-accurate computation model. In our work, the use of TLM in STARSoC platform (MPSoC platform) design refers to a set of abstraction levels quoted in [14].

2.2. STARSoC TLM

In Figure 3, all levels belong to the TLM levels except the first level. We provide below a brief description of all these levels.

(1) The first model is a *specification model* which is described by a parallel process (c program) without any architecture details.

(2) The second model is the *PE-assembly model*, implemented by using ISSs (OR1Ksim) which communicates through interprocess communication. We chose an implementation of IPC, called PIPE, for its capacity of data and command transfer.

(3) The third model is the *bus-arbitration model*. In this level, we have added two parameters: address (for memory access) and bus arbiter (for bus access), also by using IPC. In this level, STARSoC is a time approximate computation. In each clock cycle, the ISS performs one instruction. In our work, this level can also be called an instruction accurate execution model.

(4) The fourth model is the *cycle-accurate computation model*. In this model, each ISS is cycle accurate.

The advantage of this model is that it allows designers to exploit the platform at earlier stages of the design flow.

3. Proposed Architecture

The target architecture used in this work is a multiprocessor system-on-chip (MPSoC) platform, whose communications are performed via a shared memory, as shown in Figure 1.

Our reference platform consists in integrating several ISSs wrapped under the systemC wrapper interface, the TLM wishbone BUS model, private memories (associated with each ISSs), and a shared memory used for communication between the ISSs. Because we have a small set of processors

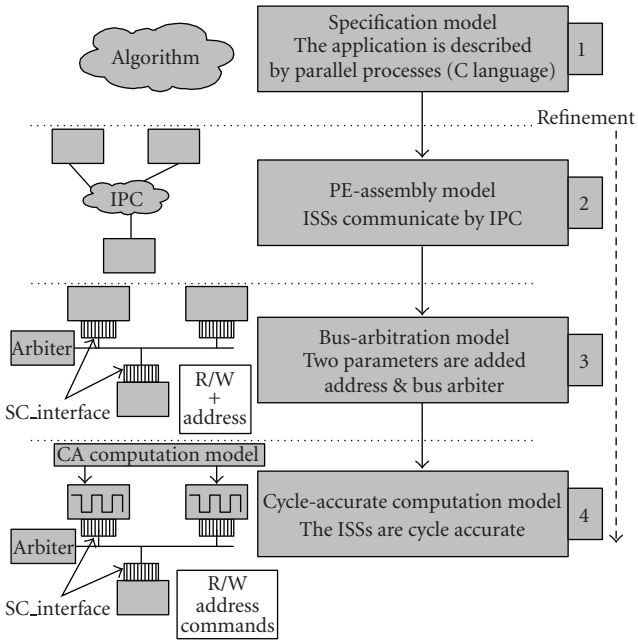


Figure 3: STARSoC platform at different abstraction levels.

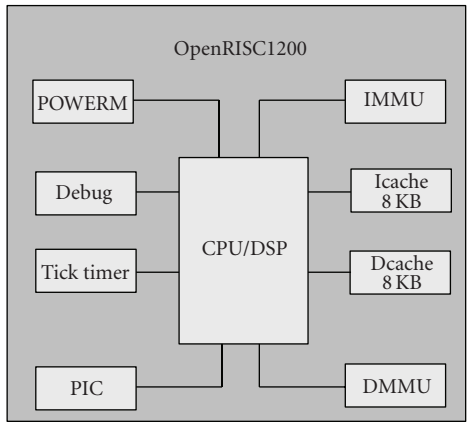


Figure 4: ISS unit architecture.

in this work, we chose a shared bus as the interconnection model instead of network-on-chip (NoC) and shared memory as the communication model instead of message passing.

The ISSs used in our platform are derived from the open source simulator of the OpenRISC processor (OR1200), written in C and called OR1Ksim (shown in Figure 4). Each ISS contains its own cache memory, private local memory, and the minimum set of units required to perform basic functionality (see Figure 4). These ISSs can provide detailed functional information, such as register values, the program execution time, and other timing information, used among them to carry out the comparison between three abstraction levels: instruction accurate level (IA) which corresponds to the third level in Figure 3, cycle-accurate (CA) level which corresponds to fourth level in Figure 3 (owned to TLM), and RTL during simulation.

```
                or1ksim\cpu\or32\insnset.c
INSTRUCTION (1_sb)
  {
    . . .
  if (add_s)
    {
      . . .
      write (fd_sb_data, &PARAM1, strlen (&PARAM1));
      write (fd_sb_add, &add, strlen (&add));
      . . .
    }
    . . .
  }
. . .
INSTRUCTION (1_1bz)
  {
    . . .
  if (add_1)
    {
      . . .
      write (fd_1bz_add, &add, strlen (&add));
      read (fd_1bz_data, &data, strlen (&data))
      . . .
    }
    . . .
  }
```

Algorithm 1: Example of an IPC Call from the ISS.

The TLM wishbone bus model is used to connect the ISSs with the rest of the system. These ISSs are executed simultaneously and share the memory address space used for inter processor communication.

The platform is entirely implemented in systemC language, except for ISSs which are wrapped under systemC. All these components are connected by the TLM wishbone BUS model, also implemented in systemC. The TLM bus model is based on the basic Wishbone communication protocol functionality at a high-abstraction level. It executes the Wishbone bus transaction without timing accuracy or pin accuracy. Besides the private memories which are associated with each ISS, our platform includes one main shared memory used for communication.

In order to wrap ISS under systemC, a systemC wrapper interface is added to the ISS C model. This process involves defining a systemC module and adding input/output ports that correspond to the ISS input/output arguments which we have implemented in the load/store functions from the OR1k CPU directory (see Algorithm 1). This can be done by using interprocess communication as shown in Figure 5. The systemC wrapper interface is made sensitive to a positive clock edge. At every positive clock edge, the systemC wrapper interface calls the corresponding C function inside the ISS via IPC, such as PIPE (used in our example) and Sockets [19]. SystemC wrappers and ISSs C models are both run as distinct Linux processes.

The communication between the components (software-software/software-hardware) can be effected using this

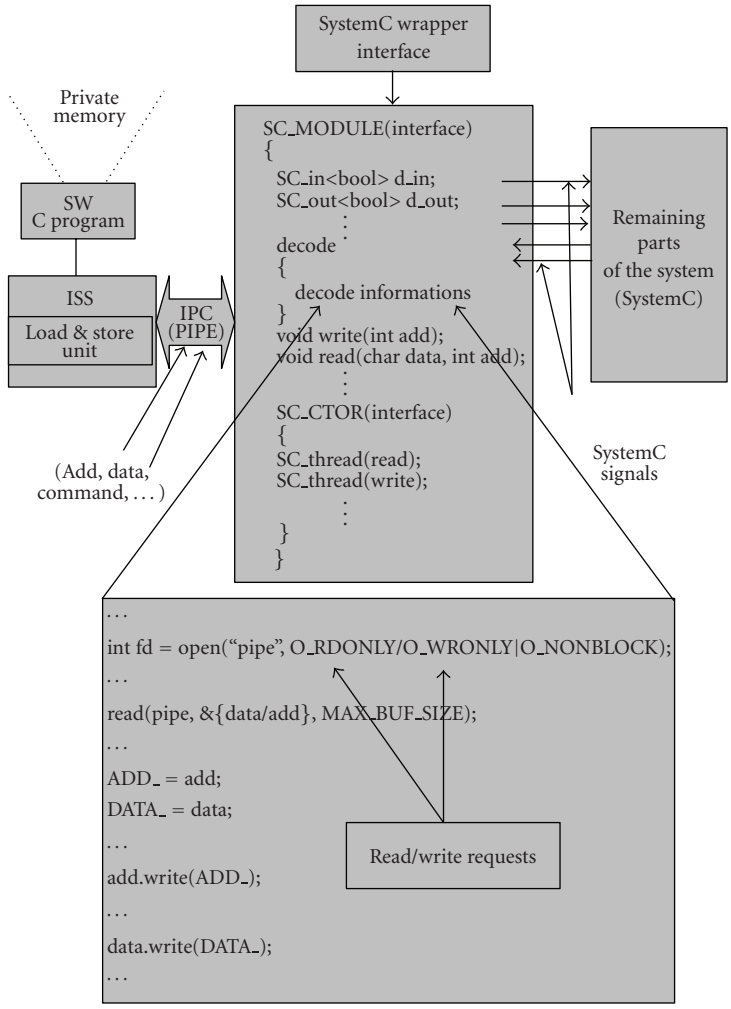


Figure 5: SystemC ISS wrapper interface.

shared memory, through systemC signals. In this case, the multiprocessing environment synchronization must be ensured by using the semaphore. Our simulation environment is configurable and allows specification of several parameters, such as the type of interconnect (Wishbone,..., etc.), the number of processing elements, and memory parameters.

In our example, we have used two ISSs, the first one writes data on the shared memory, the second ISS reads the data written by the first one (see Figure 6), taking into account the synchronization between them.

At the beginning, we wrapped each ISS under SystemC. At every clock cycle, the ISSs execute one instruction, in order to perform instruction accurate simulations. The pipeline effects are not considered at all. In this case, we have a systemC instruction accurate simulation. We then refine the simulation to a cycle-accurate simulation; we fully simulate the processor pipeline and we take into account the number of cycles necessary for each instruction (processor pipeline stage, memory access,..., etc.). We have in this case a full systemC cycle-accurate computation simulation.

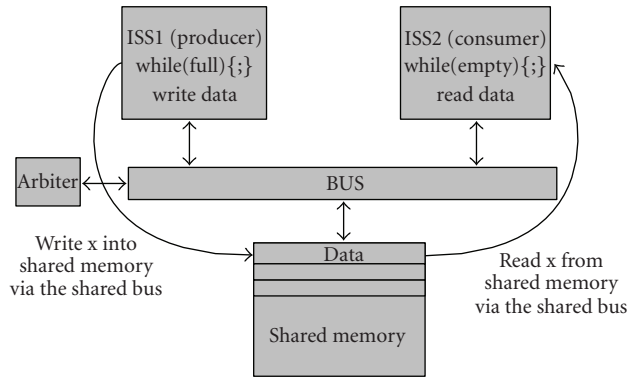


Figure 6: Our communication model using shared memory.

4. Bus Architecture

Most SoC designs are based on hardware blocks connected together with bus signals which are classified as groups of data, address, and control links. Several companies provide

the following SoC bus architectures so that designers can easily integrate the IP blocks into a single silicon chip: Core Connect, AMBA, CoreFrame, Wishbone, and Silicon Backplane Network. Our architecture platform is designed around the Wishbone bus, whose architecture was developed by Silicore Corporation, Minn, USA [20]. In 2002, Silicore placed the bus specification into the public domain via OpenCores [13] which is an organization that promotes the development of open IP cores. The Wishbone bus architecture is very simple since it defines only one bus protocol. However, the Wishbone bus architecture supports various features depending on the desired bus operations: multiple masters, single cycle read/write, block transfer cycles that systematically perform a set of single read cycles and/or a set of single write cycles.

However, in the case of TLM simulation, the wishbone bus protocol needs to be redefined as transaction-level ports of TLM. To the best of our knowledge, there has been no TLM implementation for the wishbone bus to date, we have only descriptions at signal level. It is, therefore, necessary to map signals into TLM transaction level ports (not pin accurate and not cycle accurate as only the computation models are cycle accurate). These models (TLM) should meet a few requirements including the following:

(i) speed,

(ii) flexibility,

(iii) the ability to model and evaluate several arbitration schemes,

(iv) clarity and ease in integrating other component models efficiently.

5. Transaction-Level Modeling of the Wishbone Bus System

We implemented a transaction-level model of the wishbone bus. This model accurately respects the wishbone bus protocol.

Since the bus is modeled as an abstract channel without including any specific details of the bus protocol, it enables faster communication simulation models.

We present all the steps in our methodology to develop a wishbone bus TLM architecture, by describing our transaction-level modeling steps. We first used function calls (performed by calling IPC functions) to model the wishbone signals and we then used systemC signals to implement the wishbone protocol.

In the first step, we modelize the behavior of each transaction-level port. For example, in the RTL handshaking protocol, a master can immediately get an *ACK_I* (bus grant signal) from the bus, after sending an STB_O (bus request signal) if the bus is ready (free). This step is implemented as the port's transaction of a master call *Check_bus_Grant()* and receives *true* as a return value. The arbiter selects a request from this master after applying an arbitration strategy, decodes the destination address, and sends the request to the slave destination. The arbiter calls read() and write() functions implemented in the slave. The slave receives the request from the arbiter, performs any required computation, the read/write operation, and optionally waits for a fixed number of cycles before sending a response back to the arbiter. The arbiter ensures eventual completion of the transaction. After that, the master (ISS) sends ADDR (address) and DATA (read/write data). The transaction is a single *word/bytes* read/write transfer and receives ACK.

In the second step, we implemented wishbone signals by systemC signals. The translation into a systemC signal is done by an *SC_Interface* module associated to each ISSs.

The TLM wishbone bus model that we created is shown in Figure 7, where the implementation of communication is less detailed than register-transfer level.

The simulation speeds were measured at both TL model and RTL. The TL model is about 300 times faster than the RT level model.

6. Communication Model

An important aspect in the design of multiprocessor systems is the exchange of data between SoC modules. Several communication methodologies are possible, such as shared memory and message passing. However, shared memory is the most common type of interprocessor communication paradigm, for multiprocessor system-on-chip (MPSoC) platforms, where a small set of processors share a common address space.

In this section, we present the model of communication developed and implemented in our platform architecture, which is based on the shared memory approach to perform data exchange between ISSs. We thus have two different types of memories.

A private memory space exists for each ISS which cannot be seen by any other ISS in the system except for the owner. We also have memory that is shared by all ISSs and used for communication.

An example is illustrated in Figure 6 to demonstrate our communication model, established between two processors via the shared memory which is used for this purpose. The platform architecture consists of two ISSs connected by a TLM Wishbone bus model. Besides private memories, a shared memory is used by the ISSs for data exchange. The program running on ISS1 (producer) deposits data needed by the program running on ISS2, into shared memory and waits until this data is read by the program on ISS2 before depositing other data and vice versa. But in this case, we have a well-known problem which is synchronization; it is a very critical issue in platforms based on shared memory communication. This problem arises due to the fact that the bus (in the case of a shared bus) as well as the memory is shared between different ISSs. On the one hand, these ISSs exchange data through this shared memory, on the other hand, these data are sent and received via the TLM shared bus. This situation generates shared resource access conflict if we have several simultaneous requests.

Thus, we need to use a simpler arbitration scheme such as round robin (RR) and the semaphore in order to ensure synchronization. In the example shown in Figure 6, two processors are involved. We used a round-robin arbitration

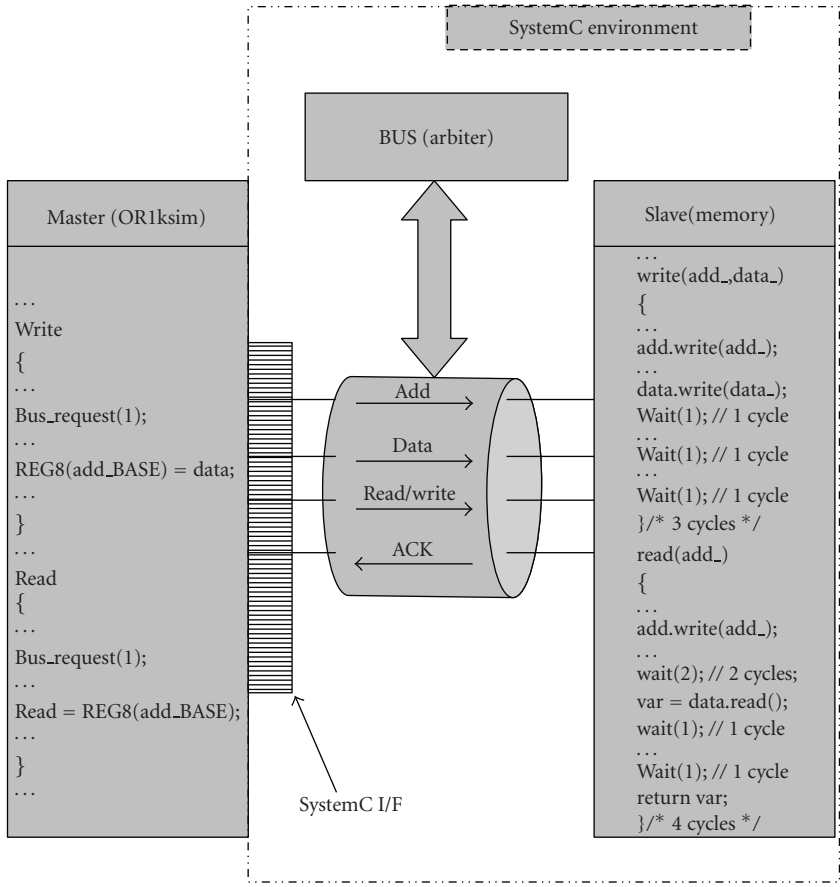


Figure 7: TLM wishbone BUS.

policy for the bus access, and we used the semaphore for shared memory access.

7. Processing Elements

The ISS used in our framework is based on OpenRISC architecture emulator, written in C and called OR1Ksim, licensed under the GNU LGPL license. The goals are to emulate 32-bit OpenRISC CPUs with a high level of abstraction capable of running real operating systems, such as NetBSD [21], RTEMS [22], eCos [23], and uClinux [24] and intended for embedded, portable, and network applications.

OpenRISC 1200 is an open source IP-core freely available from the OpenCores website [13]. This soft core is a MIPS-based 32-bit scalar RISC with Harvard microarchitecture, a 5-stage integer pipeline, virtual memory support (MMU), and basic DSP capabilities. The overview of the OR1200 architecture can be seen in Figure 4.

For this core, we have two descriptions at different abstraction levels. The first is a free open-source description written in a synthesizable Verilog code, with a low level of abstraction (RTL abstraction) verified by several functional tests and implemented into FPGAs and ASICs.

The second description is written in C code (OR1Ksim simulator) and provides several features [25]:

Table 1: Simulation time results at different abstraction levels.

	Number of iterations		
	10	100	1000
Abstraction levels	Simulations times		
Insruction accurate (IA)	0.97 (*S*)	1.5 (*S*)	8.4 (*S*)
Cycle accurate (CA)	1.38 (*S*)	2.3 (*S*)	12.32 (*S*)
RTL	3.4 (*S*)	12.56 (*S*)	62.24 (*S*)

(i) free, open source code,

(ii) high level and fast architecture simulation that allows code analysis at an earlier stage and system performance evaluation,

(iii) supports all major models of OpenCores peripheral and system controller cores,

(iv) easy addition of new peripheral models,

(v) remote debugging through a network socket with the GNU Debugger (GDB),

(vi) support for different environments (memory configurations and sizes, OR1K processor model, configuration of peripheral devices).

The tools used for compilation and debugging are the standard GNU toolchain, including the GCC compiler which

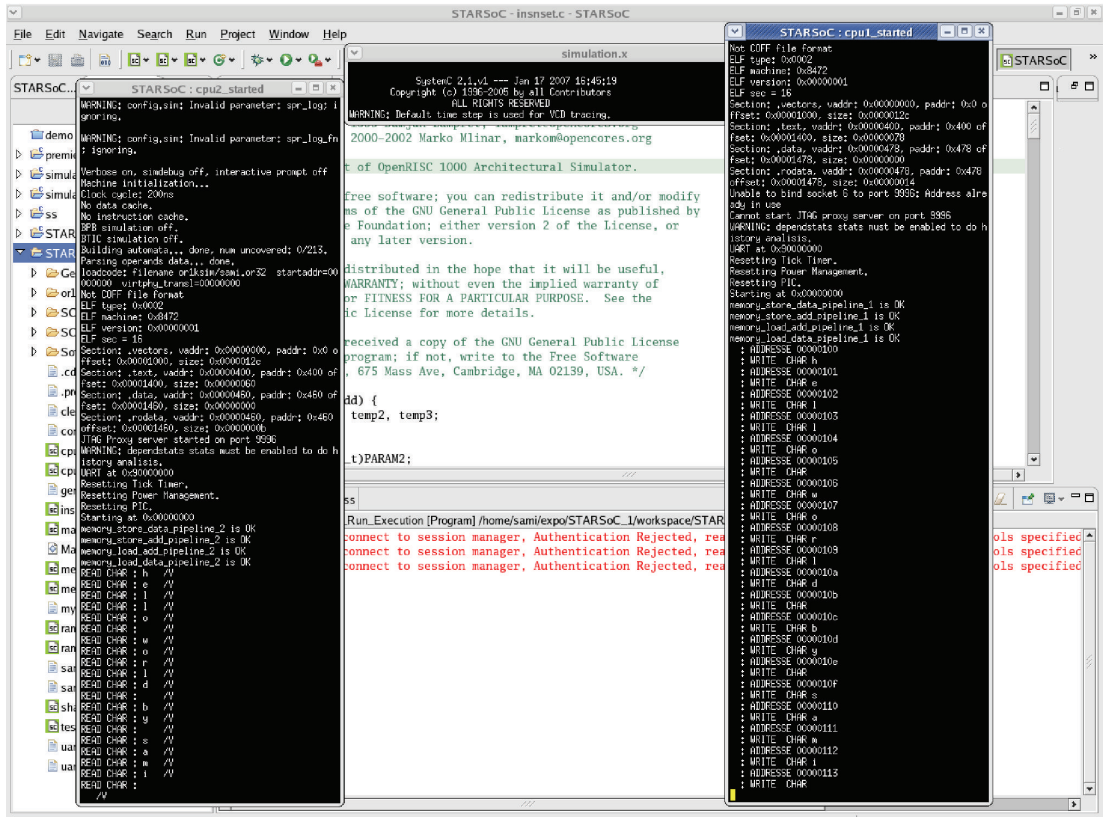


Figure 8: STARSoC IDE.

works well. The debugger is available through GDB, and it is easy with DDD which provides a more user friendly environment.

8. Results

In this section, we first describe the system architecture example used in our experiments, and we then present results that we carried out to validate the architecture generation process of STARSoC and evaluate its performances. We also compare performances among the different abstraction levels.

In our experimental example, we used a shared-memory MPSoC platform generated by a STARSoC generator tool in which multiple OpenRISC processors can be integrated and interconnected by a TLM wishbone bus model.

We performed experiments using the systemC design environment which we applied to a system consisting of two ISSs (wrapped under systemC) accessing a shared memory through a TLM bus. The architecture of the system is shown in Figure 6. All system components were specified in systemC, except for the two ISSs.

The bus arbitration mechanism is managed by the module labeled bus arbiter which is implemented by the round robin arbitration policy. In each cycle, one of the masters has the highest priority to access the shared resource. If the token-holding master does not need the bus at this cycle, the master with the next highest priority who sends a request can be granted the resource.

The interface between the bus and the ISS interface consists of four signals: a read/write signal *rw*, an address *addr*, the data *data*, and an acknowledge signal *ACK*. Similar signals exist between the bus and the shared memory thanks to which the access is synchronized by a semaphore.

ISS was built by the GNU cross-compiler (GCC version 3.4) and cross-debugger (GDB version 5.0) with the OpenRISC as a target.

The application executed by the two processors is stored in their local memory and consists of a producer-consumer (sends characters from one processor to the other) application executed in parallel and synchronized by the same clock signal. The application is loaded on the private memory of the producer core (writer) which then writes those characters into the shared memory at a given address. The consumer core (reader) reads those data from the same address. If the consumer is faster than the producer, the memory will be empty and the consumer waits until the producer writes data. Conversely, if the producer is faster, the memory will be full, and the producer waits until the consumer reads the data.

The same environment was employed for simulation on different abstraction levels. The simulation results were obtained by executing the platform on a Pentium IV at 3.0 GHz with a RAM memory size of 786 MB, based on Linux Fedora Core 3. Table 1 shows three columns 10, 100, and 1000, corresponding to the fixed number of iterations (reads or writes to the memory) of the algorithm which is performed by each ISS. The simulation times shown in

Table 1 correspond to the execution times of the programs at the different simulation abstraction levels.

Three models were generated and simulated: an instruction accurate model (bus arbitration model), a cycle-accurate system simulation model, and a synthesizable RTL model (using Verilog and simulated with ModelSim). Generally, the first two models are used to validate the software and the system architecture (they include the ISSs, bus model at transaction level, and functional models of other components, all of which use systemC models).

All model descriptions were automatically generated from the STARSoC generator tool shown in Figure 8. They have also been validated by simulation using the same testbenches.

After running the testbenches, we obtained some experimental results.

(i) The instruction accurate model is about 7 times faster than the RTL model.

(ii) It runs twice as fast as a full cycle-accurate system simulation.

(iii) Our cycle-accurate model runs five times as fast as an equivalent RTL simulation.

The simulation time analysis was used to compare the efficiency obtained from the TLM description in systemC and from the underlying RTL platform. The results reported above demonstrate the advantage of using a higher level of abstraction than RTL when carrying out architectural exploration.

9. Conclusion

The availability of a fast high level simulation makes architecture exploration possible at different abstraction levels.

In this paper, we have presented and validated our methodology for cosimulation at a high level of abstraction (TLM) within a single simulation environment based on systemC language.

Our environment is based on the use of open source ISSs C models (OR1Ksim), wrapped under systemC language by using UNIX interprocess communication.

Comparing three different abstraction levels, namely, instruction accurate level, cycle-accurate level, and RTL level (VHDL model), we have analyzed the STARSoC generated multiprocessor SoC platform.

The experimental results show that the use of systemC as a modeling language for the design of abstraction levels may significantly reduce the design validation time, enabling the development of very fast models. In addition, the simulation results at higher levels of abstraction show that there are no significant communication overheads between the ISS C model and its systemC wrapper, due to the fact that we have a small number of used cores. This model would be very useful for functional HW/SW cosimulation of large SoCs based on OpenRISC.

This motivates our choice for systemC as a system design language, dedicated to architecture exploration in our STARSoC project which is the main contribution of this work. This gives the designer the possibility of exploring the STARSoC platform at several levels, which represents a notable advantage for STARSoC design flow.

In future work, we plan to add an embedded operating system like eCos and to integrate heterogeneous IPs cores in our platform.

References

[1] P. G. Paulin, C. Pilkington, M. Langevin, et al., "Parallel programming models for a multiprocessor SoC platform applied to networking and multimedia," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 7, pp. 667–680, 2006.

[2] A. A. Jerraya, A. Bouchhima, and F. Pétrot, "Programming models and HW-SW interfaces abstraction for multiprocessor SoC," in *Proceedings of the 43rd Annual Conference on Design Automation (DAC '06)*, pp. 280–285, San Francisco, Calif, USA, July 2006.

[3] K. Hines and G. Borriello, "Dynamic communication models in embedded system co-simulation," in *Proceedings of the 34th Design Automation Conference (DAC '97)*, pp. 395–400, Anaheim, Calif, USA, June 1997.

[4] I. Petkov, P. Amblard, M. Hristov, and A. Jerraya, "Systematic design flow for fast hardware/software prototype generation from bus functional model for MPSoC," in *Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping (RSP '05)*, pp. 218–224, Montreal, Canada, June 2005.

[5] J. Jung, S. Yoo, and K. Choi, "Performance improvement of multi-processor systems cosimulation based on SW analysis," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '01)*, pp. 749–753, Munich, Germany, March 2001.

[6] Coware.Inc., N2C3, http://www.coware.com/#cowareN2C.html.

[7] Seamless, CVE, 2005, http://www.mentor.com/seamless.

[8] Open SystemC Initiative, SystemC Version 2.0, Users Guide, 2001, http://www.systemc.org/.

[9] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Legacy SystemC co-simulation of multi-processor systems-on-chip," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '02)*, pp. 494–499, Freiburg, Germany, September 2002.

[10] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "MPARM: exploring the multi-processor SoC design space with systemC," *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 41, no. 2, pp. 169–182, 2005.

[11] W. R. Stevens, *UNIX Network Programming, Volume 2: Interprocess Communications*, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[12] N. Saint-Jean, G. Sassatelli, P. Benoit, L. Torres, and M. Robert, "HS-scale: a hardware-software scalable MP-SOC architecture for embedded systems," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07)*, pp. 21–28, Porto Alegre, Brazil, March 2007.

[13] OpenCores, http://www.opencores.org/projects/or1k.

[14] L. Gai and D. Gajski, "Transaction level modeling: an overview," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '03)*, pp. 19–24, Newport Beach, Calif, USA, October 2003.

[15] A. Samahi and E.-B. Bourennane, "Automated integration and communication synthesis of reconfigurable MPSoC platform," in *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS '07)*, pp. 379–385, Edinburgh, UK, August 2007.

[16] M. B. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented FPGA computing in the Streams-C high level language," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, pp. 49–56, Napa Valley, Calif, USA, April 2000.

[17] Coware, http://www.coware.com/.

[18] Cadence, http://www.cadence.com/.

[19] W. R. Stevens, B. Fenner, and A. M. Rudoff, *Unix Network Programming, Volume 1: The Sockets Networking API*, Addison Wesley, Reading, Mass, USA, 3rd edition, 2003.

[20] Silicore, http://www.pldworld.com/_hdl/2/_ip/-silicore.net/wishbone.htm.

[21] NetBSD, http://www.netbsd.org/.

[22] RTEMS, http://www.rtems.com/RTEMS.

[23] eCos, http://ecos.sourceware.org/.

[24] uClinux, http://www.uclinux.org/.

[25] OpenCores, http://pkgsrc.se/emulators/or1ksim.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 403086, 11 pages
doi:10.1155/2008/403086

Research Article

A Game-Theoretic Approach for Run-Time Distributed Optimization on MP-SoC

Diego Puschini,[1] Fabien Clermidy,[1] Pascal Benoit,[2] Gilles Sassatelli,[2] and Lionel Torres[2]

[1] *CEA /Leti-Minatec, 17 rue des Martyrs, 38054 Grenoble Cedex 9, France*
[2] *LIRMM, University of Montpellier 2/CNRS UMR 5506, 161 rue Ada, 34392 Montpellier Cedex 5, France*

Correspondence should be addressed to Fabien Clermidy, fabien.clermidy@cea.fr

Received 29 February 2008; Accepted 12 August 2008

Recommended by Michael Hubner

With forecasted hundreds of processing elements (PEs), future embedded systems will be able to handle multiple applications with very diverse running constraints. Systems will integrate distributed decision capabilities. In order to control the power and temperature, dynamic voltage frequency scalings (DVFSs) are applied at PE level. At system level, it implies to dynamically manage the different voltage/frequency couples of each tile to obtain a global optimization. This paper introduces a scalable multiobjective approach based on game theory, which adjusts at run-time the frequency of each PE. It aims at reducing the tile temperature while maintaining the synchronization between application tasks. Results show that the proposed run-time algorithm requires an average of 20 calculation cycles to find the solution for a 100-processor platform and reaches equivalent performances when comparing with an offline method. Temperature reductions of about 23% were achieved on a demonstrative test-case.

1. Introduction

"MP-SoC is not just coming: it has arrived" [1]. Intel's Tera Scale computing research program [2] is one illustration of today's reality: a prototype is composed of 80 processors interconnected with a network-on-a-chip (NoC) [3, 4]. In a near future, an increasing number reconfigurable processing elements on a single chip is forecasted, leading to new challenges for system-level designers. From technological side, the variability between dies requires chip-by-chip optimization. Offline methods are no more possible when hundred of processors are integrated on advanced technologies. Thus, on-chip distributed techniques are required to adjust the parameters of each chip. On the applicative side, MP-SoC platforms will support several applications, including someone unknown at design time. Then, run-time adaptability is a requirement to optimize applicative parameters.

In this paper, we consider MP-SoC platforms integrating hundreds of reconfigurable processing elements (PEs) as shown in Figure 1, each includes processors, memories, and peripherals. It is assumed that each PE embeds the required components to locally adjust its parameters. For instance, dynamic voltage/frequency scaling (DVFS) tunes the voltage/frequency couple of each tile. In this context, an important issue is how to manage the tradeoff between the performance achieved and the die temperature which is an indirect indicator of the power consumption. It is a multiobjective optimization problem [5]: how to solve the power and temperature management for hot-spot reduction [6] taking into account the performance control through task synchronization for a given application with functional dependencies [7].

The main contribution of this paper is the use of the game theory [8] as a model to dynamically optimize MP-SoC platforms in a distributed way. A run time game-theoretic method to choose the frequency for each PE in MP-SoC platforms in order to optimize the circuit temperature taking into account the task synchronization has been presented in [9]. In this article, this technique is reviewed. A statistical analysis regarding the algorithm convergence and scalability is studied and a demonstrative test-case is presented.

1.1. Problem Formulation

Consider an MP-SoC architecture composed of several reconfigurable processing elements (PEs). It is assumed that future MP-SoC will integrate a large number of PEs leading to the choice of a distributed control architecture. Each PE integrates devices such as processors, memories, peripherals, and dynamic voltage/frequency scaling (DVFS). The choice of distributed DVFS is justified in [10, 11] while in [12, 13], a whole demonstrator is presented.

It is assumed that synchronous PEs are connected by an asynchronous 2D-mesh NoC as in [14]. The interconnect system offers the required bandwidth and latency for the targeted applications, as well as the ability of individual frequency selection.

Consider an application composed of n tasks T_i and connections as illustrated by the example described in Figure 2(a). It is assumed that the application is supported by the MP-SoC and it is mapped on the platform by a given mechanism. At application level, the functional dependencies between tasks lead to adjust the frequency of each PE in order to guarantee the task synchronization [7].

At physical level, the frequency choice is influenced by the temperature metric. In this paper, we consider a first-order approximation that the temperature of a given PE is affected by its neighbors [6, 15], as shown in Figure 2(b). The two constraints meet when the application is mapped on the MP-SoC (Figure 2(c)). A tradeoff between the synchronization and the temperature metrics has to be solved. It is a multiobjective optimization problem we solve in this paper.

1.2. Paper Organization

The paper is organized as follows. Section 2 presents some related works regarding dynamic and static optimizations for embedded reconfigurable systems. The basic formalisms, definitions, and notations of game theory are presented in Section 3. In Section 4, we present the multiobjective optimization notation and we formulate our task synchronization and temperature models. Based on discussed models and using the preliminary presented theory, a game-theoretic optimization algorithm is proposed. Simulations were performed on several scenarios to study the feasibility of this approach in the MP-SoC context. Section 5 analyzes the dynamic behavior and its optimization quality. Finally, a demonstrative test case is evaluated in Section 6.

2. Related Works

Several optimization methods are used to get the best tradeoff between system metrics for a given architecture. For instance, designers try to obtain the best performance on power consumption ratio (MIPS/mW). Optimization may be applied at different stages, either at design time (static optimization) or at run time (dynamic optimization), or both, as described in the following paragraphs.

Several works propose some static optimization techniques for given metrics. In [16], the authors developed a new framework based on integer linear programming (ILP) solvers and constraint programming to solve at design time the task allocation/scheduling problem. In this domain, there are also several significant contributions such as [17, 18] for workload optimization and [19] for energy savings.

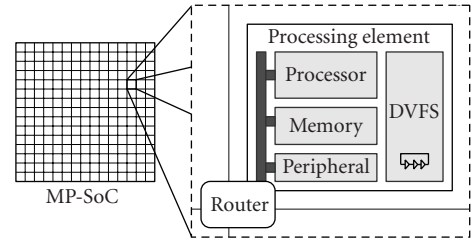


Figure 1: MP-SoC with hundreds of processing elements connected by an NoC. Each PE integrates a DVFS.

More recently, there is a growing interest for run-time optimization. When dynamic methods are considered, most of the proposed approaches are global techniques: the optimization decisions are taken considering the whole system. In [20], heuristics for optimal task placement are discussed in an NoC-based heterogeneous MP-SoC. There are several approaches where tasks are moved in order to balance computation workload and homogeneously dissipate the power, for example, in [6, 21]. The slow reaction time, the requirement of unused tiles, and the memory or the important number of data transfer between tiles can limit the use of these techniques for certain applications. Moreover, the implementation of these techniques is limited to functionally homogeneous arrays.

In [22, 23], authors propose a design time Pareto exploration and characterization combined with run-time management [24]. In [15], a convex optimization method is used in a 2-phase algorithm that allows frequency assignment on an MP-SoC controlling hot-spots. These approaches become prohibitive as the applications should be completely characterized at design time for an efficient implementation. In [25, 26], voltage and frequency are chosen at run time by centralized mechanisms. These solutions do not scale with the number of PEs and then do not well match for future multiprocessor platforms [1].

Our contribution is based on the use of game theory for dynamic optimization. Game theory has been widely used in other domains such as economy, sociology, and biology. In the VLSI context, in [27], game theory has been firstly used for circuit synthesis. To the best of our knowledge, it has never been applied to run-time optimization of embedded systems.

3. Game Theory

The objective of this section is to introduce the notations used in the game theory and the common solution known as the Nash equilibrium (NE) [28]. Game theory was introduced by Von Neumann and Morgenstern [8]; it can be viewed as a branch of applied mathematics to study interactions among rational individuals or decision makers.

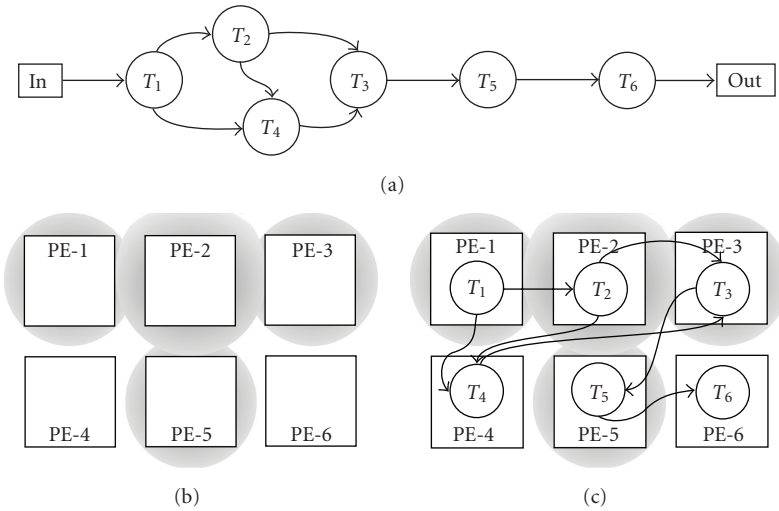


Figure 2: (a) An application example with 6 tasks: each task has to be processed at a given frequency to guarantee the synchronization between logical neighbors. (b) The MP-SoC composed of 6 PEs: the temperature of each PE is affected by physical neighbors. (c) The application is mapped on the MP-SoC: applicative and physical requirements are merged on a two-metric optimization problem.

3.1. Noncooperative Games

A game is a scenario with several players interacting by actions and consequences [29]. Basically, the players, or decision makers, individually choose how they act, resulting in consequences or outcomes. Each decision maker tries to maximize its own outcome according to its preferences, leading to a global optimization. Since the decision is made without need to the cooperation of other players, the game is called *noncooperative*.

Mathematically, such a game in *normal form* (normal form is the way in which the game is described) with n players is described as

$$\Gamma = \{N, S_i, u_i\}, \quad \forall i \in N, \tag{1}$$

where $N = \{1, \ldots, n\}$ is the set of players; S_i is the set of actions for players i; and u_i is a function describing its outcomes. The discrete set of actions or *strategies* of player i is defined as

$$S_i = \{s_{i1}, s_{i2}, \ldots, s_{im_i}\}, \tag{2}$$

where m_i is the number of possible actions for this player. The outcome of player i is represented by a score: the higher the score, the nearer the optimal point. Because of dependencies with other players, the score or *utility* is a function of choices from current player and choices from other players:

$$u_i : S_1 \times S_2 \times \cdots \times S_n \longrightarrow \Re. \tag{3}$$

For MP-SoC platforms, the choice of the game type to be applied is driven by the complexity, for low-cost implementation, and the number of steps of the game for the run-time feature. For these reasons, the *noncooperative normal-form simultaneous repeated game* model has been chosen.

3.2. The Nash Equilibrium

Nash proved that each n-player noncooperative game has at least one equilibrium point, known as Nash equilibrium (NE). It can be defined by *pure strategies* or by *mixed strategies*. In pure strategies, the solutions are obtained by allowing only one action per player. On the other side, in mixed strategies, the solution is chosen in a set of actions, each played with a given probability [28]. For pure strategies, an equilibrium point is defined as follows.

Definition 1. For a given game $\Gamma = \{N, S_i, u_i\}$, a solution

$$S^* = \{s_1^* \in S_1,\ s_2^* \in S_2, \ldots, s_n^* \in S_n\} \tag{4}$$

is a Nash equilibrium if for all $i \in N$,

$$u_i\{s_1^*, \ldots, s_i^*, \ldots, s_n^*\} \geq u_i\{s_1^*, \ldots, s_i, \ldots, s_n^*\}, \tag{5}$$

where s_i^* is the Nash equilibrium strategy for player i.

Note that in an NE, players cannot improve their outcomes by unilaterally changing their strategies, indicating then that it is at least a local maximum.

4. Game-Theoretic Multiobjective Optimization

In this section, a new algorithm based on game theory is proposed to solve the multiobjective optimization issue at run time. The mathematical description of multiobjective optimization problems is formulated. Then, the system-level metrics are discussed and finally, the game-theoretic model and the distributed optimization algorithm are proposed.

4.1. Multiobjective Optimization

The multiobjective optimization problem is defined as how to find a vector $X = (x_1, x_2, \ldots, x_q)$ that optimizes a vector

function $\vec{O}$ whose elements represent l-normalized objective functions $O_j(X)$. These functions are mathematical descriptions of performance criteria, usually in conflict. That is,

$$\vec{O}(X) = (O_1(X), O_2(X), \ldots, O_l(X)), \tag{6}$$

where $\vec{O}(X)$ produces solutions in an l-dimension space. The problem is converted in a single objective optimization problem thanks to the linear combination of the metrics [5]:

$$O'(X) = w_1 O_1 + w_2 O_2 + \cdots + w_l O_l, \tag{7}$$

where $\sum_{j=1}^{l} w_j = 1$ and $w_j \geq 0$ for $j = 1, \ldots, l$. The weights w_j represent the importance of each metric in the combination $O'(X)$.

This principle combined with the game theory can be used to optimize the task synchronization when minimizing the temperature of the blocks by the run-time selection of the PE frequency. Thus, (7) can be written for each PE of the MP-SoC as

$$O_i = w_{syn,i} O_{syn,i}(F) + w_{temp,i} O_{temp,i}(F), \tag{8}$$

where $F = (f_1, f_2, \ldots, f_n)$ is the frequency of each PE, $O_{syn,i}(F)$ represents the synchronization function, and $O_{temp,i}(F)$ is the temperature metric for each PE. Finally, $w_{syn,i}$ and $w_{temp,i}$ are the relative importance of each metric in the optimization process.

4.2. System-Level Metrics

4.2.1. Task Synchronization Model

The application synchronization problem is defined as the choice of the best working frequency for each processor. In [30], authors try to equalize the input and outputs rates of application nodes in order to maintain the processor rate. Based on this principle, we propose a simple task synchronization model based on the load assigned to each PE and its frequency.

Assume that block i supports the task T_i and transfers the generated data to block j, which is running the task T_j. They are synchronized when they are working at the same mean performance, that is, the data produced by block i are entirely consumed by block j. Otherwise, if block i is faster than block j, some data produced by the first block are not immediately consumed by the second one. On the contrary, if block i is slower than j, the second one will have undesirable idle cycles. In both cases, they are not synchronized.

Following this principle, we define the synchronism between two blocks by the following equation:

$$A_{syn_{i,j}} = -\left| \frac{\alpha_i f_i}{L_i} - \frac{\alpha_j f_j}{L_j} \right|, \tag{9}$$

where f_i and f_j are the frequencies of blocks i and j, L_i and L_j are their respective computation loads. Parameters α_i and α_j are considered to normalize heterogeneous PEs. In the rest of the paper, they are assumed to be $\alpha_i = \alpha_j = 1$ for homogeneous MP-SoC. The result of this equation is zero when the blocks are synchronized, and a negative value otherwise.

Equation (9) can now be used to build the first metric for each PE as follows:

$$O_{syn,i} = -\frac{1}{k_i}\left[A_{syn_{i,1}} \;\; A_{syn_{i,2}} \;\; \ldots \;\; A_{syn_{i,n}}\right] C_i, \tag{10}$$

where k_i is the number of connections and C_i is the connectivity vector of task T_i for the application:

$$C_i = \left[c_{i1} \;\; c_{i2} \;\; \ldots \;\; c_{in}\right]^T, \tag{11}$$

where $c_{ij} = 1$ if task i is connected to task j and $c_{ij} = 0$ otherwise.

Finally, the problem of global task synchronization results in the choice of the frequency f_i that maximizes $O_{syn,i}$ for each PE.

4.2.2. Temperature Model

Considering a fixed voltage, the temperature of a block depends on the power consumption and the effect induced by other blocks. On a simplified model, we neglect the static consumption. We consider the dynamic power consumption of the PE as $P_i = \beta_i f_i V^2$ with f_i the clock frequency, V the supply voltage and β_i a given circuit constant.

Following [31], the transfer thermal resistance R_{ij} of PE_i with respect to PE_j is

$$R_{ij} = \frac{\Delta T_{ij}}{\Delta P_j}, \tag{12}$$

where ΔT_{ij} is PE_i temperature rise due to the power ΔP_j dissipated by PE_j.

The temperature of each PE in an n-processor array is calculated by

$$T_{\mathrm{PE}_i} = R_{i1} P_1 + R_{i2} P_2 + \cdots + R_{in} P_n, \tag{13}$$

where $P_1, \ldots, P_n$ are the power consumptions of the n-PEs. For simplification purposes, it is assumed in this work that the PEs are only affected by the power dissipated by the nearest neighbors. In a regular 2-dimension mesh array, it means the PEs located at the north, south, east, and west positions.

Finally, expression (13) is reformulated for each PE in order to express the reduction of temperature as an optimization metric:

$$O_{temp,i} = -R_i\left[P_1 \;\; P_2 \;\; \ldots \;\; P_n\right]^T \tag{14}$$

with R_i a vector containing the transfer thermal resistances R_{ij} of PEs affecting the temperature of PE_i.

4.3. Game-Theoretic Method

The tradeoff problem discussed in Section 1.1 is modeled as a game (as explained in Section 3) with multiple objectives (Section 4.1). The components of such a game are identified and an algorithm is proposed to solve the formulated problem.

```
Require: u_i, MyStgy, OtherStgies
Ensure: NewStgy
    NewStgy ⇐ MyStgy
    for all Stgy do
        if u_i(Stgy, OtherStgies) > u_i(NewStgy, OtherStgies)
        then
            NewStgy ⇐ Stgy
        else
            NewStgy ⇐ NewStgy
        end if
    end for
```

Algorithm 1: UnilaterallyMax(player-i).

4.3.1. Game Model

As stated in Section 3.1, a game Γ is composed of players (N), a set of actions per player (S_i), and the outcomes (u_i). For the given tradeoff problem, the PEs will be considered as the player set. The tradeoff variable is the clock frequency of each PE. Thus, the set of actions of each player is each possible frequency f_i. Setting the frequency step allows to deduce the strategy space S_i. For example, with a 5 MHz step into the bounds of 100 to 300 MHz, the strategy space will be as follows:

$$S_i = \{s_{i1} = 100,\ s_{i2} = 105, \ldots, s_{i41} = 300\}. \quad (15)$$

The outcome or utility function u_i is described by the linearized version of the multiobjective optimization problem, that is, by using (8):

$$u_i = w_{syn,i}O_{syn,i}(s_1, \ldots, s_n) + w_{temp,i}O_{temp,i}(s_1, \ldots, s_n), \quad (16)$$

where $O_{syn,i}$ and $O_{temp,i}$ are, respectively, the objective functions of (10) and (14). $w_{syn,i}$ and $w_{temp,i}$ are the synchronization and temperature weights of the optimization for block i.

4.3.2. Distributed Algorithm

The method used to select the choice is defined as *unilateral maximization*: each player chooses the action maximizing its own utility function. The choices of other players are considered as given parameters. The maximization can be performed by running Algorithm 1 for each PE, where u_i is the utility function of PE i, $MyStgy$ is its last chosen strategy, and *OtherStgies* are the strategies chosen by other players in the previous cycle. Note that this code implements the utility maximization by comparing the outcomes of all possible solutions per player.

The code is embedded in each PE and is simultaneously executed at run time, allowing the parallel distributed optimization. The period between two executions is defined as *game cycle*. It is assumed that before the next game cycle, all PEs will end the current execution of the maximization process.

```
for all GameCycle do
    for all i do
        NewStgy ⇐ UnilaterallyMax(player-i)
    end for
    Stgy ⇐ NewStgy
end for
```

Algorithm 2: GameKernel.

In order to analyze the global behavior, the parallel launching of Algorithm 1 is simulated by Algorithm 2. All PEs launch the unilateral maximization at the same time and taking into account the last known choice of other players. Note that once a new application is mapped or the application load changed, the game-theoretic algorithm will adjust the PE frequencies.

5. Results

The results presented in this section include the characterization of the algorithm behavior. Two aspects are analyzed. From one side, since the objective of this work is to provide a *run-time* optimization mechanism, the dynamic response of the algorithm has major interest: the convergence speed of the system is studied. On the other side, in order to characterize the quality of the solution, it has to be compared with an existing optimization method.

In this section, these two aspects are analyzed from a statistical point of view. Firstly, the exploration methodology is discussed. Secondly, the convergence results are presented regarding the system scalability. Then, an exploration of the impact of the objective weights on the dynamic response is presented. This section ends with a comparison of the optimization quality of the game-theoretic solution with an offline method.

5.1. Methodology

The game complexity is defined by the number of PEs needed by the application (i.e., the number of players),

```
for Size = 4 to 100 do
    for Connectivity = 2 to Size −1 do
        for Scenario = 1 to 1000 do
            Generate random application
            Run reference optimization analysis
            Save results
        end for
    end for
end for
```

Algorithm 3: Statistical simulation script.

the application connectivity (i.e., the interaction between players), and the multiobjective function u_i (i.e., the utility function). In order to study the scalability of our technique, the application size is explored in a range of 4 to 100 processors. For each evaluated case, the maximum application connectivity is explored from 2 links per task to full connectivity ($n - 1$, where n is the number of tasks). The third aspect, the utility function, is explored by changing the objective weights w_{syn} between 0 and 1 and w_{temp} between 1 and 0.

A first analysis of the convergence speed is done by setting the objective weights to $w_{syn} = 1$ and $w_{temp} = 0$ and then repeating several different scenarios. In order to obtain statistical results, each parameter combination (i.e., size and application connectivity) is simulated 1000 times. For each time, a new application graph is defined. The simulation procedure is implemented by Algorithm 3.

In a second convergence analysis, the impact of the objective weights is explored. Several random 25-task applications are evaluated for each pair w_{syn}, w_{temp}.

5.2. Convergence

The dynamic response is analyzed by measuring the convergence speed in a number of game cycles that the algorithm takes to reach a solution. Statistically, the convergence speed corresponds to a gauss distribution with mean and standard deviation depending on the game parameters (number of processors and actions). For example, Figure 3 shows the distribution results over 98000 simulations of a 100-processor platform with random applications, connections chosen between 2 per PE to full connectivity. This curve indicates that typically a 100-processor MP-SoC will find a solution in around 18 game cycles but usually in less than 40 cycles, regardless of the application.

The analysis of the mean and the standard deviation of the convergence speed are repeated for the simulation results in a range of 4 to 100 processors. Figure 4 shows the results for different sizes. The graph shows 3 regions corresponding to the concentration of 68%, 95%, and 99.7%, respectively, the first, second, and third standard deviations. For instance, 99.7% of applications will converge in less than 32 game cycles in a 36-processor system. Note that the convergence speed decreases in $O(\log(n))$ when the platform

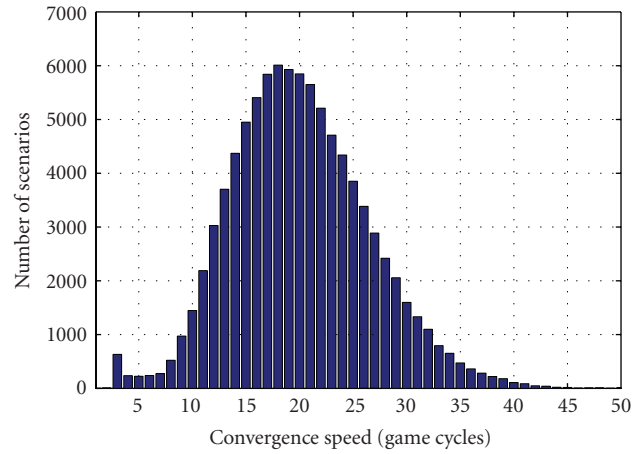


Figure 3: Convergence speed distribution for a 100-processor platform with $w_{syn} = 1$ and $w_{temp} = 0$: the average speed is 18 game cycles.

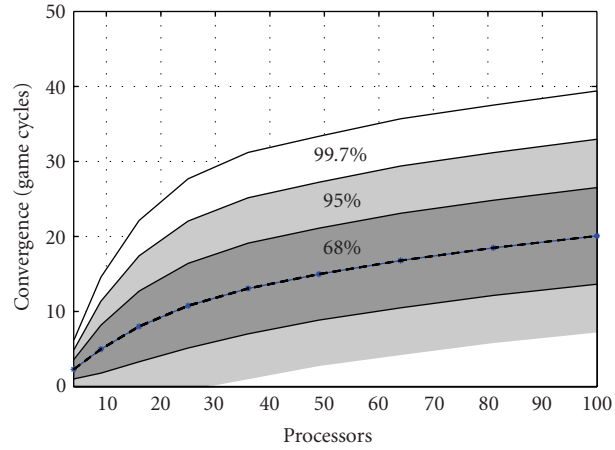


Figure 4: Convergence speed distribution from 4 to 100 processors with $w_{syn} = 1$ and $w_{temp} = 0$. The mean and standard deviations are indicated.

size augments, showing that the algorithm perfectly scales with the number of processors.

A game cycle is composed by two phases. The first one consists of the communication of the decisions of all PEs. The second phase is the maximization of the utility function. A generic 8051-microcontroller has been used to illustrate a game cycle duration. It takes around 400 clock cycles to process the algorithm or 800 nanoseconds considering a 500 MHz frequency. On the other side, the communication latency in a 2D mesh NoC is estimated as follows. The longest path in an n-processor system is $2\sqrt{n} - 1$ hops. It is assumed that there is no conflict in the interconnect. Considering that in the asynchronous NoC the equivalent node latency is around one cycle of the given 500 MHz clock, the maximum estimated communication delay is 38 nanoseconds per game cycle for a 100-processor MP-SoC. Table 1 summarizes the speed results measured in a number of game cycles and in the equivalent estimated times.

Table 1: Convergence speeds.

	Average*		Worst*	
Platform size	Game cycles	Time [μs]	Game cycles	Time [μs]
9 PEs	5	4.10	17	12.10
16 PEs	8	6.51	23	18.72
25 PEs	10	9.00	28	22.90
36 PEs	13	10.69	32	26.30
49 PEs	15	12.39	34	28.08
64 PEs	16	13.28	36	29.88
81 PEs	18	15.01	37	30.86
100 PEs	20	16.76	40	33.52

* From probabilistic analysis.

Table 2: Test case solutions.

	Configuration 1 ($w_{syn} = 1,\ w_{temp} = 0$)		Configuration 2 ($w_{syn} = 0.75,\ w_{temp} = 0.25$)		Configuration 3 ($w_{syn} = 0.5,\ w_{temp} = 0.5$)		Configuration 4 ($w_{syn} = 0.25,\ w_{temp} = 0.75$)	
PE	GT Algo. [MHz]	Matlab [MHz]	GT Algo. [MHz]	Matlab [MHz]	GT Algo. [MHz]	Matlab [MHz]	GT Algo. [MHz]	Matlab [MHz]
1	160	164	160	156	160	152	155	152
2	110	114	110	107	110	100	105	100
3	160	164	155	149	155	132	150	105
4	100	103	100	100	100	100	100	100
5	225	231	190	210	140	185	140	147
6	160	164	135	149	100	132	100	105
Conver.	17 game cycles	—	17 game cycles	—	4 game cycles	—	4 game cycles	—

From the statistical simulations, it is observed that 94% of evaluated scenarios converge to a solution. The other 6% cases do not converge to a unique solution but they present oscillations between two or more frequencies for each PE. It is assumed that in these conflictive cases an external mechanism such as task migration [32, 33] is used to solve the problem.

5.3. Impact of Weights

In order to study the impact of the objective weights on the convergence speed, 50 000 simulations are performed over a 5×5-processor array where each PE drives its frequency between 100 and 200 MHz with a step of 10 MHz. Applications are defined with random loads and connections as in the previous experience. For each simulation, a new application is randomly generated. The results are shown in Figure 5. The y-axis represents the number of scenarios found for each convergence speed from x-axis and for a given metric weight. The z-axis explores the synchronization weight w_{syn} from 0 to 1, corresponding to w_{temp} from 1 down to 0.

The results show that the convergence speed augments with w_{syn}, meaning that it depends on the metric complexity. For instance, for the trivial case of $w_{syn} = 0$, all the frequencies are driven to the minimum value in the first game cycle. It is due to the improvement of the temperature alone, without taking care of the task synchronization. On the other side, for $w_{syn} = 1$ the system presents the slowest convergence speed, with an average of 10 game cycles.

5.4. Optimization Quality

In the game-theoretic model, each player tries to optimize its outcome by maximizing the utility u_i. From a global point of view, the outcome is described by

$$U : \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \longmapsto \begin{bmatrix} u_1(s_1, s_2, \ldots, s_n) \\ u_2(s_1, s_2, \ldots, s_n) \\ \vdots \\ u_n(s_1, s_2, \ldots, s_n) \end{bmatrix}. \tag{17}$$

The global optimization problem is then formulated as

$$\max_{S} \left\{ \min_{u_i} \{U(S)\} \right\}, \quad s_{li} \le s_i \le s_{ui} \tag{18}$$

where s_{li} and s_{ui} are the lower and upper bounds of the strategy space of player i, that is, the minimum and maximum frequencies. This formulation is known as the *minimax problem* [34].

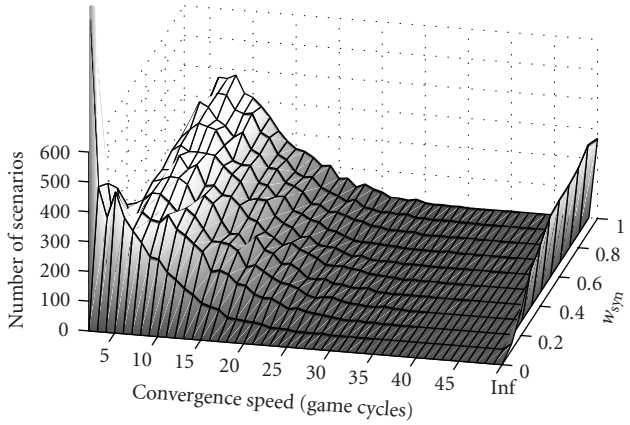


Figure 5: Weights exploration for a 5×5-processor array. The x-axis represents the number of scenarios over 50 000 simulations. The y-axis shows the convergence speed in game cycles, while the z-axis explores the weight of the synchronization metric w_{syn}. The inf label on the y-axis denotes those cases which do not converge.

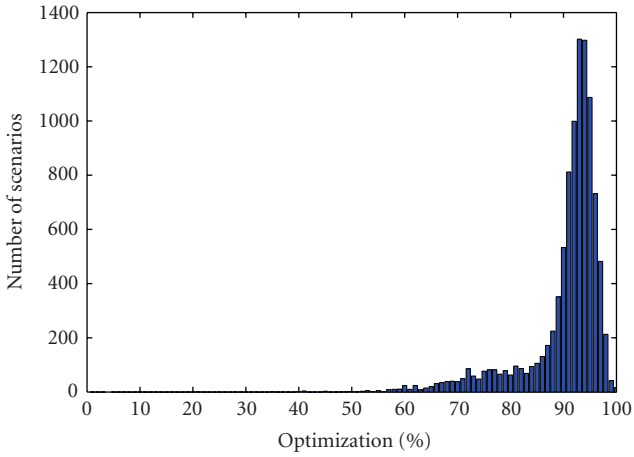


Figure 6: Optimization percentage distribution compared to Matlab minimax solution: an average of 89% and a peak of 93% of optimization are reached.

Using Matlab minimax solution, worst and best bounds are found for each simulated scenario. The game-theoretic solution is then positioned between these two references allowing the characterization of the quality of the found solution. A total of 10 000 simulations of the procedure used in Section 5.2 are analyzed, calculating the optimization percentage achieved in each case. The results are presented as a distribution curve in Figure 6. As it is shown, the distribution shows an average at 89% while it presents a peak at 93%. The results are concentrated between 58 and 98%. Note that these results are obtained in few game cycles, for instance, less than 40 for a 100-processor MP-SoC as was explained in Section 5.2. On the contrary, the Matlab minimax algorithm takes between some seconds and few minutes to calculate each solution on a nowadays desktop PC.

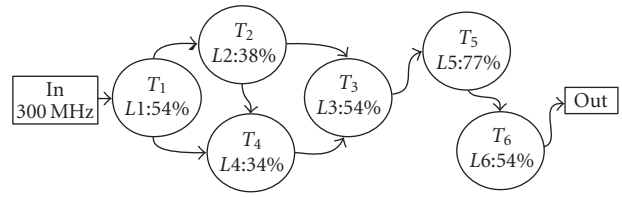


Figure 7: Application example composed of 6 tasks mapped on a 3 × 2-PE array.

6. Test Case

For clarity of the demonstration, a very simple test case composed of a 6-task application (Figure 7) mapped on a 3 × 2-PE array (Figure 2(c)) has been chosen. Each PE is able to adjust its own frequency between 100 and 300 MHz with a step of 5 MHz. The task synchronization is modeled as in Section 4.2.1 while the temperatures of PEs are calculated as in Section 4.2.2. The transfer thermal resistances are arbitrarily assumed to be $R_{ii} = 0.7$ and $R_{ij} = 0.3$ (the real values are dependent on the used technology). The two metrics are combined as in (16) by the weights w_{syn} for the task synchronism and w_{temp} for the PE temperature.

Four configurations were evaluated. The first one describes a system which is only interested in optimizing the task synchronization, that is, $w_{syn} = 1$ and $w_{temp} = 0$. In the context of this work, this configuration is used as the reference to calculate the temperature reduction achieved by the other configurations. The second one expresses a scenario where the synchronization represents 75% of the optimization importance; while only 25% is for the temperature minimization ($w_{syn} = 0.75$ and $w_{temp} = 0.25$). The third configuration defines a case with equal interest for each metric ($w_{syn} = 0.5$ and $w_{temp} = 0.5$), while the last case gives only 25% of importance for the synchronization ($w_{syn} = 0.25$ and $w_{temp} = 0.75$). For simplicity purpose, these values are arbitrary chosen to be the same for all PEs. Nevertheless, PEs may have different constraints. For example, a central PE may have more interest in temperature reduction than a border one to avoid hot-spots.

In order to measure the quality of the found solution, the same optimization issue is modeled as the minimax problem presented in Section 5.4 and solved using Matlab. The results are compared to the game-theoretic solution.

Figure 8 shows the evolution of the game-theoretic algorithm for the second configuration ($w_{syn} = 0.75$ and $w_{temp} = 0.25$). Each graph of the figure shows in solid lines the evolution of the chosen frequency. In this example, processor 1 takes only two game cycles to reach a stable solution, while processor 2 needs 3 cycles, and processor 3 needs 4 game cycles. Processor 4 has chosen the lowest frequency from the beginning; finally, processors 5 and 6 are the slowest that need 17 and 16 game cycles, respectively, to reach the solution. In this example, the game reaches the NE in 17 cycles. In dashed lines, Figure 8 shows the optimal solutions found with Matlab. After few game cycles, the game-theoretic solution converges to an NE close to the Matlab solution.

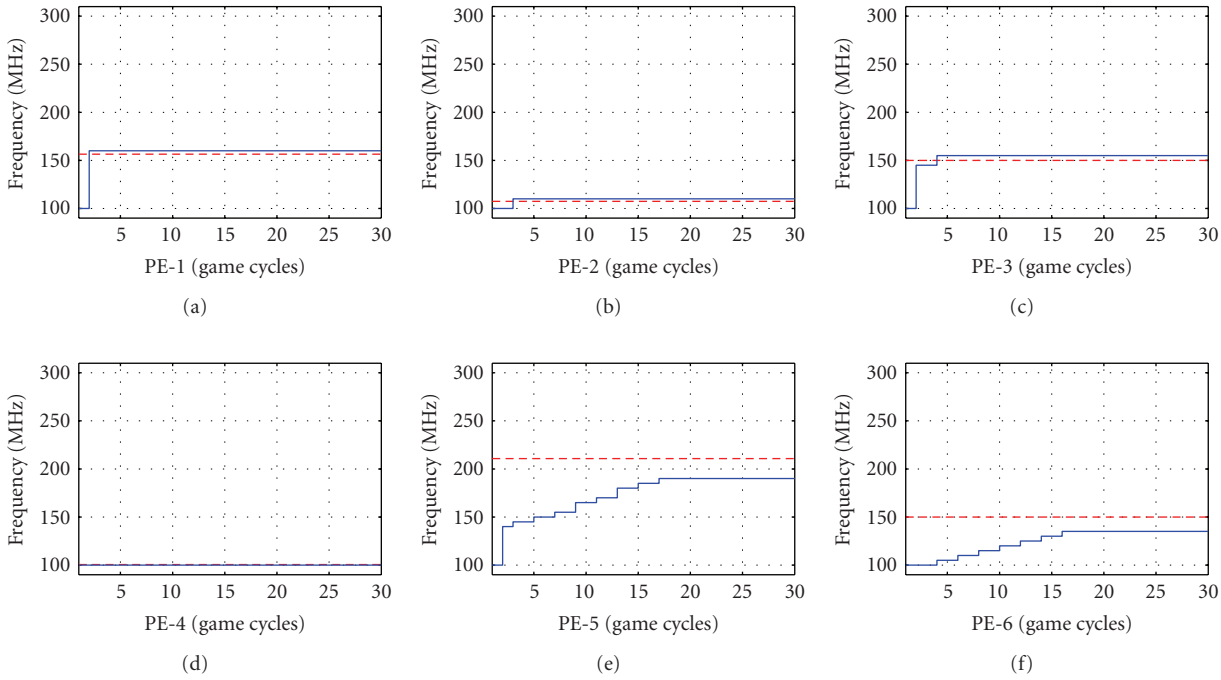


Figure 8: Dynamic response of a 3×2 processor array for a given 6-task application with $w_{syn} = 0.75$ and $w_{temp} = 0.25$. The solid line shows the evolution of the chosen frequency for each PE. The NE is defined by $f_1 = 160$, $f_2 = 110$, $f_3 = 155$, $f_4 = 100$, $f_5 = 190$, and $f_6 = 135$ MHz. The dashed line is the reference solution calculated by Matlab: $f_1 = 156$, $f_2 = 107$, $f_3 = 149$, $f_4 = 100$, $f_5 = 210$, and $f_6 = 149$ MHz.

Table 3: Test case temperature profiles.

	Configuration 1 ($w_{syn} = 1$, $w_{temp} = 0$)		Configuration 2 ($w_{syn} = 0.75$, $w_{temp} = 0.25$)		Configuration 3 ($w_{syn} = 0.5$, $w_{temp} = 0.5$)		Configuration 4 ($w_{syn} = 0.25$, $w_{temp} = 0.75$)	
PE	GT Algo.	Matlab	GT Algo.	Matlab	GT Algo.	Matlab	GT Algo.	Matlab
1	35.0°C	35.9°C	35.0°C	34.2°C	35.0°C	33.2°C	34.0°C	33.2°C
2	48.1°C	49.5°C	45.7°C	45.8°C	42.7°C	42.1°C	41.4°C	38.2°C
3	38.6°C	39.6°C	36.4°C	36.2°C	34.3°C	32.4°C	33.3°C	7.0°C
4	37.1°C	38.1°C	35.0°C	35.9°C	32.0°C	34.2°C	31.7°C	31.9°C
5	53.7°C	55.2°C	47.3°C	50.7°C	38.2°C	45.8°C	37.9°C	38.8°C
6	45.5°C	46.6°C	39.6°C	42.4°C	31.7°C	37.5°C	31.4°C	29.8°C
T. Avg	43.0°C	44.1°C	39.8°C	40.9°C	35.6°C	37.5°C	34.9°C	33.1°C
T. Gain	—	—	7.95%	7.99%	23.94%	17.63%	23.02%	33.11%

Table 4: Test case synchronization profiles.

	Configuration 1 ($w_{syn} = 1$, $w_{temp} = 0$)		Configuration 2 ($w_{syn} = 0.75$, $w_{temp} = 0.25$)		Configuration 3 ($w_{syn} = 0.5$, $w_{temp} = 0.5$)		Configuration 4 ($w_{syn} = 0.25$, $w_{temp} = 0.75$)	
PE	GT Algo.	Matlab	GT Algo.	Matlab	GT Algo.	Matlab	GT Algo.	Matlab
1	97.14%	98.16%	97.14%	94.68%	97.14%	88.86%	94.85%	88.86%
2	95.11%	97.23%	96.28%	93.18%	96.28%	81.81%	94.87%	68.43%
3	97.30%	98.31%	89.72%	94.42%	76.32%	85.02%	75.49%	64.51%
4	98.19%	99.10%	97.21%	92.77%	97.21%	81.26%	93.30%	71.21%
5	96.30%	96.65%	80.33%	97.11%	50.92%	96.22%	55.11%	96.80%
6	96.30%	96.65%	97.01%	97.11%	96.96%	96.22%	96.96%	96.80%
Avg	96.72%	97.69%	92.96%	94.88%	85.80%	88.23%	85.10%	81.10%

Tables 2, 3, and 4 summarize the results of the four evaluated configurations. Table 2 lists the frequencies found by the game-theoretic algorithm and by Matlab. Note that in all cases, the solutions found by the game-theoretic algorithm are close to those found with Matlab. The convergence speed of the game-theoretic solution for each configuration is also highlighted. Table 3 presents the resulting temperature of each PE, the average temperature of the entire system, and the gain achieved by configurations 2, 3, and 4 compared to configuration 1. These results show up to 23% of average temperature gain, depending on w_{temp} value. Note that these reductions are obtained in few game cycles, making this technique able to manage the parameters at run time.

In addition, Table 4 lists the improvement percentage of task synchronization. The task synchronization of each PE has been calculated by using expression (10) for a nominal case where all PEs work at 200 MHz. The synchronization improvements for the game-theoretic and Matlab optimizations are calculated for each configuration with respect to the nominal case. The results are listed in Table 4, showing for configuration 1 ($w_{syn} = 1$, $w_{temp} = 0$) an average improvement of 96.72% for the game-theoretic optimization and 97.69% for the Matlab one compared to the nominal frequency set. Note that for configurations 1, 2, and 3 Matab obtains better synchronizations at higher temperatures than the game-theoretic algorithm. On the contrary, for configuration 4, Matlab obtains worse synchronizations at lower temperatures compared to our algorithm.

Finally, the maximum, average, and minimum PE temperatures for the game-theoretic algorithm are represented in Figure 9. The results show more uniform temperature distribution when w_{temp} rises. Configuration 1 presents 19°C of difference between maximum and minimum (44% of the average), while configuration 4 only shows 10°C (28% of the average temperature). The run-time game-theoretic method has not only reduced the average temperature but also the peaks or hot spots.

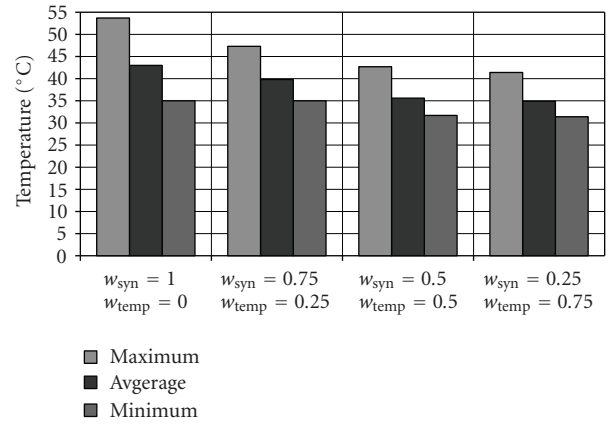


Figure 9: Maximum (Max), average (Avg), and minimum (Min) temperatures for the game-theoretic solution for 4 weight configurations.

7. Conclusion

In this paper, we have presented a novel run-time technique based on the game theory. We have discussed the optimization of multiple objectives on embedded reconfigurable systems. We have proposed an algorithm that optimizes the temperature profile while maintaining the task synchronization. Compared to other approaches, our technique assumes a complete distributed multiprocessor system able to take decisions at run time.

The results have shown that our method scales with the number of processor without excessive convergence times. For a 100-processor platform, our technique has required an average of 20 calculation cycles to reach the solution, that is, about 16 μs when using the 8051 microcontroller at 500 MHz. The few calculation cycles needed to converge make this technique able to optimize metrics at run time on massively parallel embedded systems.

We have measured that the achieved optimization is about 89% in average compared to a global offline method. The evaluated test case has showed that our algorithm can achieve reductions of up to 23% in the temperature profile.

References

[1] G. Martin, "Overview of the MPSoC design challenge," in *Proceedings of the 43rd Annual Conference on Design Automation (DAC '06)*, pp. 274–279, ACM, San Francisco, Calif, USA, July 2006.

[2] http://techresearch.intel.com/articles/Tera-Scale/1421.htm.

[3] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Annual Conference on Design Automation (DAC '01)*, pp. 684–689, Las Vegas, Nev, USA, June 2001.

[4] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[5] J. L. Cohon, *Multiobjective Programming and Planning*, Academic Press, New York, NY, USA, 1978.

[6] G. M. Link and N. Vijaykrishnan, "Hotspot prevention through runtime reconfiguration in network-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '05)*, vol. 1, pp. 648–649, IEEE Computer Society, Munich, Germany, March 2005.

[7] M. Ruggiero, A. Acquaviva, D. Bertozzi, and L. Benini, "Application-specific power-aware workload allocation for voltage scalable MPSoC platforms," in *Proceedings of the IEEE International Conference on Computer Design (ICCD '05)*, pp. 87–93, IEEE Computer Society, San Jose, Calif, USA, October 2005.

[8] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, Princeton University Press, Princeton, NJ, USA, 1944.

[9] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, "Temperature-aware distributed run-time optimization on MP-SoC using game theory," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '08)*, pp. 375–380, IEEE Computer Society, Montpellier, France, April 2008.

[10] J. Donald and M. Martonosi, "Techniques for multicore thermal management: classification and new exploration," in *Proceedings of the 33rd International Symposium on Computer*

Architecture (ISCA '06), pp. 78–88, Boston, Mass, USA, June 2006.

[11] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, "Voltage-frequency island partitioning for GALS-based networks-on-chip," in *Proceedings of the 44th Annual Conference on Design Automation (DAC '07)*, pp. 110–115, ACM, San Diego, Calif, USA, June 2007.

[12] E. Beigné, F. Clermidy, S. Miermont, and P. Vivet, "Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC," in *Proceedings of the 2nd ACM/IEEE International Symposium on Networks-on-Chip (NOCS '08)*, pp. 129–138, Newcastle upon Tyne, UK, April 2008.

[13] E. Beigné, F. Clermidy, S. Miermont, et al., "A fully integrated power supply unit for fine grain dvfs and leakage control validated on low-voltage srams," in *Proceeding of the 34th European Solid-State Circuits Conference (ESSCIRC '08)*, Edinburg, UK, September 2008.

[14] D. Lattard, E. Beigné, C. Bernard, et al., "A telecom baseband circuit based on an asynchronous network-on-chip," in *Proceedings of the 54th IEEE International Solid-State Circuits Conference (ISSCC '07)*, pp. 258–601, San Francisco, Calif, USA, February 2007.

[15] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. De Micheli, "Temperature-aware processor frequency assignment for MPSoCs using convex optimization," in *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '07)*, pp. 111–116, ACM, Salzburg, Austria, September-October 2007.

[16] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, "Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '06)*, vol. 1, pp. 3–8, Munich, Germany, March 2006.

[17] M. Kandemir and G. Chen, "Locality-aware process scheduling for embedded MPSoCs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '05)*, vol. 2, pp. 870–875, IEEE Computer Society, Munich, Germany, March 2005.

[18] F. Li and M. Kandemir, "Locality-conscious workload assignment for array-based computations in MPSOC architectures," in *Proceedings of the 42nd Annual Conference on Design Automation (DAC '05)*, pp. 95–100, ACM, Anaheim, Calif, USA, June 2005.

[19] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '04)*, vol. 1, pp. 234–239, IEEE Computer Society, Paris, France, February 2004.

[20] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs," in *Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP '07)*, pp. 34–40, IEEE Computer Society, Porto alegre, Brazil, May 2007.

[21] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*, pp. 1659–1664, EDA Consortium, Nice, France, April 2007.

[22] Ch. Ykman-Couvreur, E. Brockmeyer, V. Nollet, T. Marescaux, F. Catthoor, and H. Corporaal, "Design-time application exploration for MP-SoC customized run-time management," in *Proceedings of the International Symposium on System-on-Chip (SoC '05)*, pp. 66–69, Tampere, Finland, November 2005.

[23] Ch. Ykman-Couvreur, V. Nollet, T. Marescaux, E. Brockmeyer, F. Catthoor, and H. Corporaal, "Pareto-based application specification for MP-SoC customized run-time management," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS '06)*, pp. 78–84, Samos, Greece, July 2006.

[24] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal, "Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management," in *Proceedings of the International Symposium on System-on-Chip (SOC '06)*, pp. 195–198, Tampere, Finland, November 2006.

[25] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, vol. 35, pp. 89–102, ACM, Banff, Canada, December 2001.

[26] B. Knerr, M. Holzer, and M. Rupp, "Task scheduling for power optimisation of multi frequency synchronous data flow graphs," in *Proceedings of the 18th Symposium on Integrated Circuits and Systems Design (SBCCI '05)*, pp. 50–55, ACM, Florianopolis, Brazil, September 2005.

[27] N. Hanchate and N. Ranganathan, "Simultaneous interconnect delay and crosstalk noise optimization through gate sizing using game theory," *IEEE Transactions on Computers*, vol. 55, no. 8, pp. 1011–1023, 2006.

[28] J. Nash, "Non-cooperative games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[29] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*, MIT Press, Cambridge Mass, USA, 1994.

[30] K. Niyogi and D. Marculescu, "Speed and voltage selection for GALS systems based on voltage/frequency islands," in *Proceedings of the Conference on Asia South Pacific Design Automation (ASP-DAC '05)*, pp. 292–297, ACM, Shanghai, China, January 2005.

[31] W. Hung, C. Addo-Ouaye, T. Theocharides, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Thermal-aware IP virtualization and placement for networks-on-chip architecture," in *Proceedings of the IEEE International Conference on Computer Design (ICCD '04)*, pp. 430–437, IEEE Computer Society, San Jose, Calif, USA, October 2004.

[32] N. Saint-Jean, P. Benoit, G. Sassatelli, L. Torres, and M. Robert, "Application case studies on HS-scale, a MP-SOC for embbeded systems," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS '07)*, pp. 88–95, Samos, Greece, July 2007.

[33] N. Saint-Jean, G. Sassatelli, P. Benoit, L. Torres, and M. Robert, "HS-scale: a hardware-software scalable MP-SOC architecture for embedded systems," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07)*, pp. 21–28, IEEE Computer Society, Porto Alegre, Brazil, March 2007.

[34] J. W. Herrmann, "A genetic algorithm for minimax optimization problems," in *Proceedings of the Congress on Evolutionary Computation (CEC '99)*, vol. 2, pp. 1099–1103, IEEE Press, Washington, DC, USA, July 1999.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 180216, 14 pages
doi:10.1155/2008/180216

Research Article

An Embedded Reconfigurable IP Core with Variable Grain Logic Cell Architecture

Motoki Amagasaki, Ryoichi Yamaguchi, Masahiro Koga, Masahiro Iida, and Toshinori Sueyoshi

Graduate School of Science and Technology, Kumamoto University, 2-39-1 Kurokami, Kumamoto 860-8555, Japan

Correspondence should be addressed to Motoki Amagasaki, amagasaki@cs.kumamoto-u.ac.jp

Received 1 March 2008; Revised 15 May 2008; Accepted 21 August 2008

Recommended by Philip Leong

Reconfigurable logic devices (RLDs) are classified as the fine-grained or coarse-grained type based on their basic logic cell architecture. In general, each architecture has its own advantage. Therefore, it is difficult to achieve a balance between the operation speed and implementation area in various applications. In the present paper, we propose a variable grain logic cell (VGLC) architecture, which consists of a 4-bit ripple carry adder with configuration memory bits and develop a technology mapping tool. The key feature of the VGLC architecture is that the variable granularity is a tradeoff between coarse-grained and fine-grained types required for the implementation arithmetic and random logic, respectively. Finally, we evaluate the proposed logic cell using the newly developed technology mapping tool, which improves logic depth by 31% and reduces the number of configuration data by 55% on average, as compared to the Virtex-4 logic cell architecture.

1. Introduction

System large-scale integrated circuits (LSICs), which exhibit high performance and have high densities, are manufactured using advanced process technologies. However, their high costs are an enormous disadvantage. To respond to diversifications in market trends and frequent changes in standards, various types of products must be manufactured in low volumes, despite the fact that conventional production facilities essentially target high volumes of only a few types of LSICs. It follows that the cost benefit of each chip is poor because of the increase in both the nonrecurring expense (NRE) and design cost.

Hence, a reconfigurable logic device (RLD), which has circuit programmability, is applied to embedded systems as a hardware intellectual property (IP) core. Due to its flexibility, it is possible for designs to be implemented in the shortest turn-around time from specification to implementation. In particular, there is a possibility that design complexity and power distribution problems can be solved using a dynamic reconfigurable processor. It becomes necessary to adapt the frequently changing market cycles, such as that of mobile phones.

However, conventional RLDs, which are commercial field-programmable gate arrays (FPGAs), cannot achieve efficient implementation. Therefore, the chip area and power consumption increase. Because embedded systems, in particular, require a small area and low power consumption, the above-mentioned parameters are critical for these products. If we consider the quality of the actual product, conventional FPGAs are not satisfactory in terms of performance or function.

We have studied a reconfigurable logic architecture that has both flexibility and high performance [1–4] as a reconfigurable IP core. Our goal is specification circuits rather than large circuits, such as stand-alone commercial FPGA. Figure 1 shows the usage of reconfigurable IP cores as follows: (a) bug fixing after fabrication process, (b) avoidance of high volume, (c) version upgrade, and (d) multiple mode/specification. In the present paper, we propose a variable grain logic cell (VGLC) architecture that can change the computational granularity corresponding to

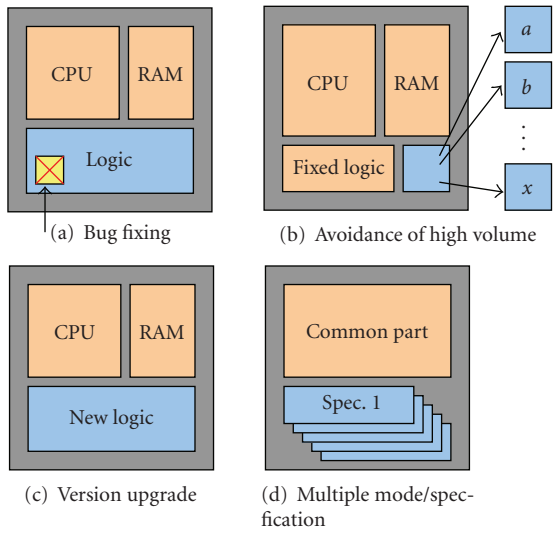


Figure 1: The use of a VGLC as a reconfigurable logic core.

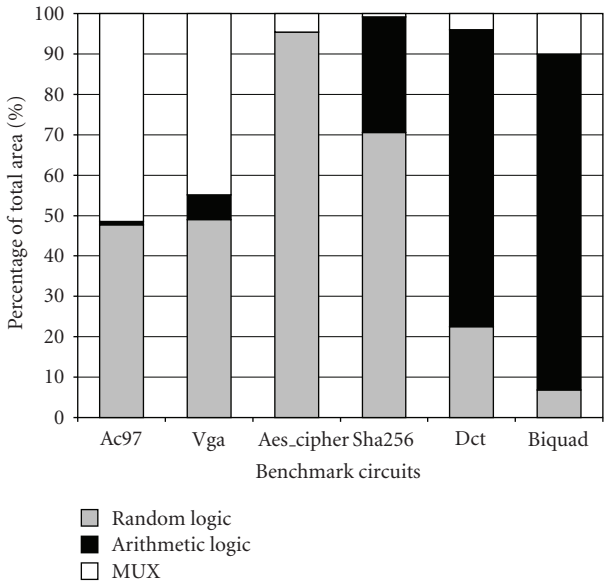


Figure 2: Synthesis results of six benchmark circuits.

the application. The novel architecture, which is based on a 4-bit ripple carry adder that includes configuration memory bits, offers a tradeoff between coarse and fine granularity and can be used for efficient mapping in an application. We also evaluate the implementation efficiency using the newly developed technology mapping tool.

The remainder of the present paper is organized as follows. Related research is described in Section 2. Section 3 introduces issues related to implementation efficiency in RLDs. Section 4 describes the VGLC architecture. Section 5 describes the technology mapping method. Section 6 describes the evaluation process and method. Section 7 presents a discussion of the results obtained herein. Finally, the conclusions and an overview of future research are presented in Section 8.

2. Related Research

The need to optimize a general-purpose reconfigurable logic cell architecture is recognized in the field of reconfigurable IP-core design. For example, mixed-grain FPGA [5] with four 2-LUTs and carry generation logic is used to reduce critical path delay and implementation area for the DSP application domain. Mixed-grained FPGA can construct a 4-bit adder or 4-input logic using one logic cell as well as our concept. They performed implementation evaluation using hand mapping for small circuits. Reference [6] also proposes memory-oriented architecture. Wilton et al. presented a synthesizable datapath-oriented embedded FPGA [7] that is optimized for bus-based operations, which are common in signal processing. This architecture will be used to implement multiply functions, rather than a more general logic circuit. NEC electronics' DRP-1 [8], Hewlett Packard Chess architecture [9], and IPFLex's DAPDNA [10] are proposed for DSP application domain. These devices are based on the ALU structure. The function units in Chess are 4-bit ALUs, and the connections are 4-bit buses. Each ALU is adjacent to four switch boxes, and each switch box is adjacent to four ALUs. On the other hand, DPFPGA [11] and the medium-grain logic cell [12] are based on the LUT structure. In [13], the proposed logic cell includes a 6:2 compressor, which features additional fast carry chains that concatenate adjacent compressors and can be routed locally without a global routing network. Altera Stratix devices [14, 15], Xilinx Virtex-5 [16], and Kumamoto University MCNC [17] are multigrained (adaptive) LUT structures. In the Stratix-II and Stratix-III, the adaptive logic module (ALM) can be used for varying input granularity. Higuchi et al. [18] presented reconfigurable hardware (RHW), which has a 1-bit full adder and EXOR and MUX. While they presented a fast depth-constrained technology mapping algorithm for RHW as tree-structured 2-LUTs, RHW only ensures 2-input logic completely. Peter Jamieson and Jonathan Rose present shadow clusters [19]. The shadow cluster is a standard FPGA logic cluster, such as a multiplier, that shares the routing resources with the hard circuit. The key advantage of the shadow cluster concept is that the waste associated with the unused programmable routing is mitigated in hard circuit tiles because a shadow cluster always allows the programmable routing designed for hard circuits to be used.

3. Application Implementation Spectrum

3.1. Granularity

Conventional RLDs are roughly classified into fine-grained and coarse-grained architecture types based on the granularity of the basic logic block [20]. The fine-grained reconfigurable architecture is associated with lookup tables (LUTs) or multiplexers, which are essentially identical to the logic elements of FPGAs. These logic elements can efficiently implement any circuit for random logic functions. On the other hand, a typical element of the coarse-grained reconfigurable architecture is a computational datapath unit, such as the arithmetic-logic unit (ALU). This architecture

Table 1: Structures of typical RLDs.

Device	Virtex-4 (XC4VLX15) [21]	DRP-1 [8]
Logic Block structure	4-LUT × 8 Carry logic × 2 MULT AND × 8 Distributed RAM 64 bit	8-bit ALU 8-bit DMU Register file (8 bit × 16 word)
# of LBs	112,288	512
Dedicated IP	18-bit multiplier	32-bit multiplier

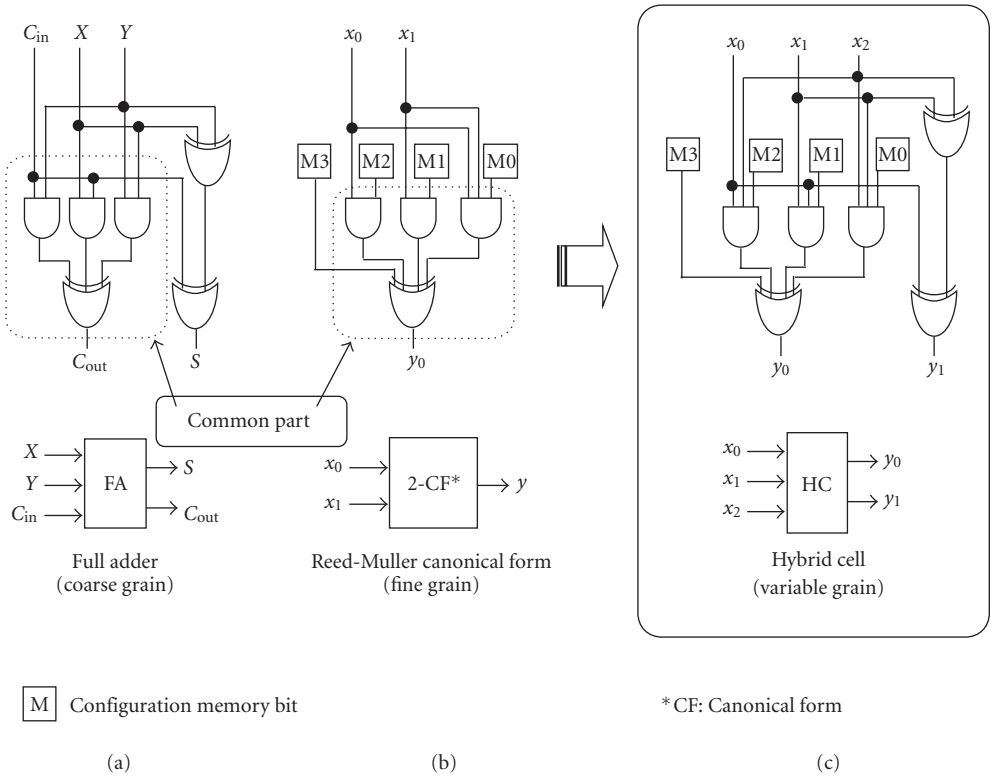


Figure 3: Basic structure of a hybrid cell.

type reduces the total amount of configuration data, thereby improving the reconfiguration overhead. NEC electronics' DRP-1 [8] and IPFLex's DAPDNA [10] are coarse-grained dynamic reconfigurable processors that are developed for the current market.

However, these RLDs can be efficiently implemented only in their specific application domains. If fine-grained RLDs incorporate a digital filter, which requires several arithmetic operations, a large circuit area will be required. On the other hand, coarse-grained RLDs have several area and delay overheads for glue logic implementation.

In order to overcome this disadvantage, traditional RLDs have dedicated circuits inside and outside the logic block. Table 1 shows the features of two typical RLDs. The Xilinx Virtex-4 (XC4VLX15) [21] comprises a carry logic, a MULT AND, and an 18-bit multiplier that is embedded as a hard IP core. On the other hand, DRP-1 has an 8-bit data management unit (DMU) as the logic unit and eight 32-bit multiplier units. However, we must carefully consider the type of functionality for dedicated circuits. If these circuits are not utilized in an application, then the space they take up on the chip is wasted. It is difficult to achieve a balance between the operation speed and area in applications.

3.2. Computational Characterization of Applications

In this section, we present a type of application domain analysis. For this purpose, we use six types of OpenCores [22] benchmark circuits that are characterized in terms of three types of application domain: the controller domain, the cryptosystem domain, and the DSP domain.

We first synthesize six industrial designs using the Synopsys Design Compiler with ASIC standard cell. Here, we extract the datapath circuit in the form of a macroblock using

Table 2: Output logic pattern of the BLE (CP = 0).

	Four-input variable		Three-input variable	
AS	*dt*	*ds*	*dt*	*ds*
0	182/65,536	24/65,536	120/256	43/256
1	230/65,536	24/65,536	148/256	43/256
Total	446/65,536 (0.68%)		206/256 (80.47%)	

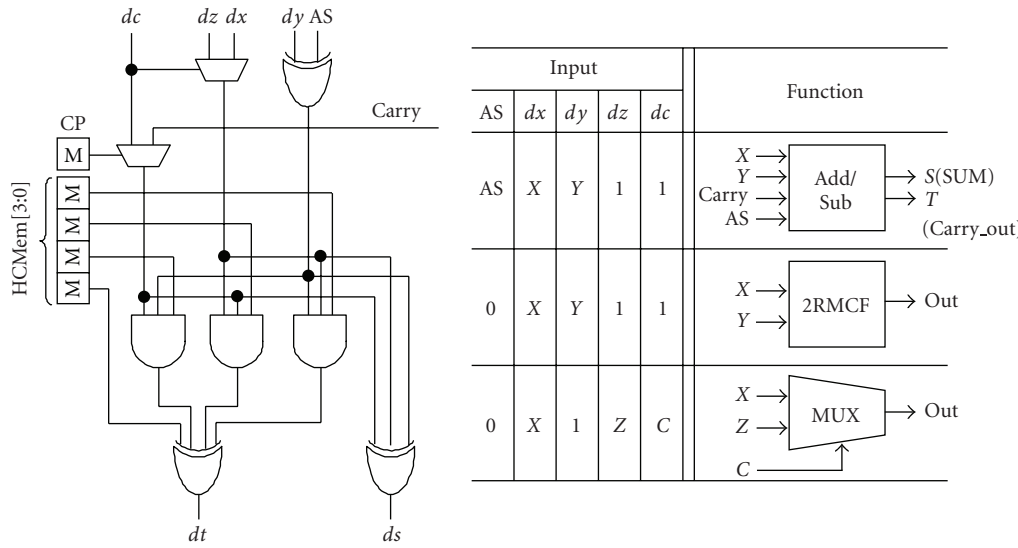


Figure 4: Components and functions of a BLE.

the Synopsys Design Ware library. This allows the extraction of various macroblocks, such as adders, multipliers, and wide-ranging MUXes. Nondetected circuits are categorized as random logic computation circuits.

We mapped these designs onto standard cells with a CMOS 0.35-μm library and generated a synthesis report containing the area information for each design. Based on the obtained results, we classified all of the design components into three main groups: random logic, arithmetic functions, and MUXes. Figure 2 shows the information obtained from the area report: Ac97 and Vga are the circuits for the control application domain, aescipher and sha256 are the circuits for the for the cryptosystem application domain, and biquad and Dct are the circuits for the DSP application domain. The graph shows the percentage of the total area taken by each function. Since flip flops are unrelated to the division of random logic or arithmetic logic, they are excluded. The ratio of random logic to arithmetic logic is not constant across different applications. In addition, this shows that the MUX occupies a large area in four circuits and has different input ranges, for example, 2:1MUX, 4:1MUX, and 8:1MUX as indicated by the synthesis report. In [5], a similar analysis and results were presented for the DSP application domain. Hence, it is necessary to effectively implement these computations in a logic block for efficiently mapping of the area. At the same time, we must consider not only the type of computation but also various input ranges.

In the following section, we propose a homogeneous architecture in which, as discussed in Section 3.2, the various functionality requirements must be fulfilled in every logic cell. The potential area overhead is avoided by maximal reuse of existing logic cell resources.

4. Proposed Logic Cell Architecture

As mentioned above, since the granularity of a conventional logic cell is fixed, it is difficult to efficiently implement circuits for all applications. This problem can be solved if the computational granularity of a logic cell unit can be changed. We propose a logic cell that has a 4-bit ripple carry adder with a configuration memory bit. In this section, we describe a hybrid cell that is used as the basic element of the VGLC. Next, we introduce a novel logic cell structure and its functions.

4.1. Hybrid Cell

The arithmetic computation is based on an adder, and the random logic is expressed efficiently in a "canonical form," such as a lookup table. Figure 3(a) shows the structure of a 1-bit full adder. Although an OR gate is generally used for the output of the full adder, an EXOR gate can also be used. Figure 3(b) shows the 2-input Reed-Muller canonical form.

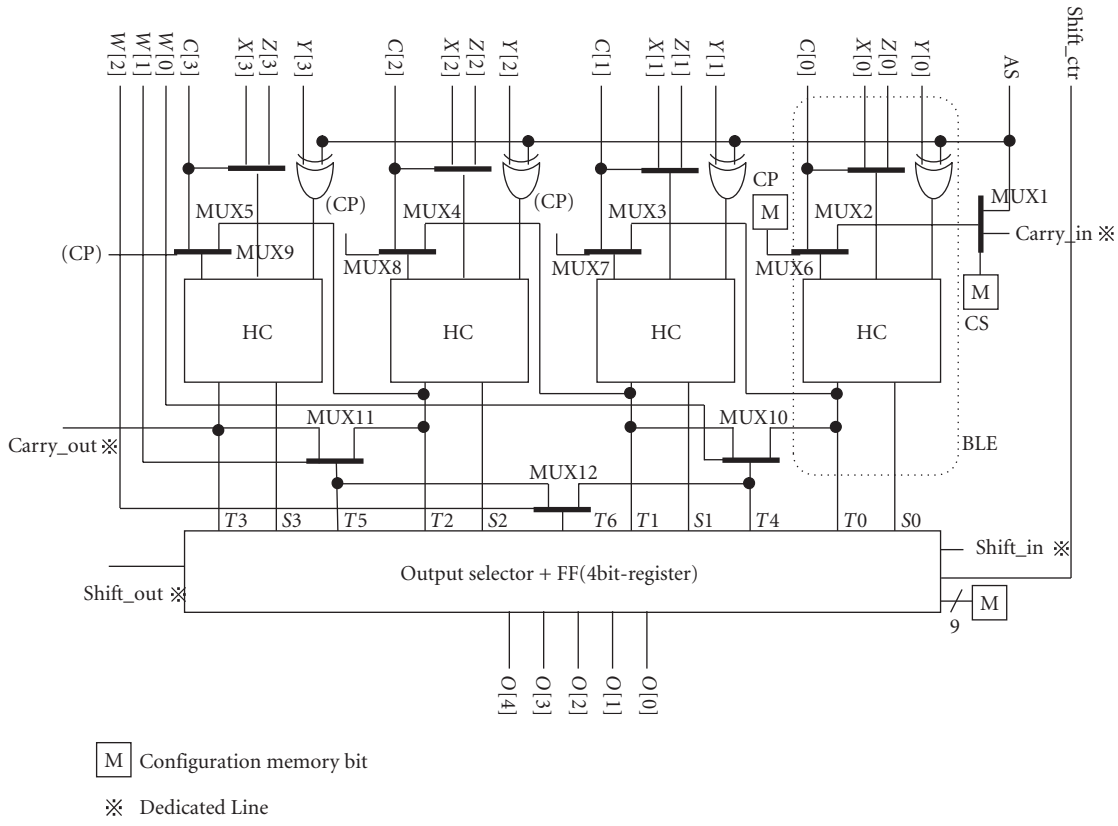


Figure 5: Variable grain logic cell architecture.

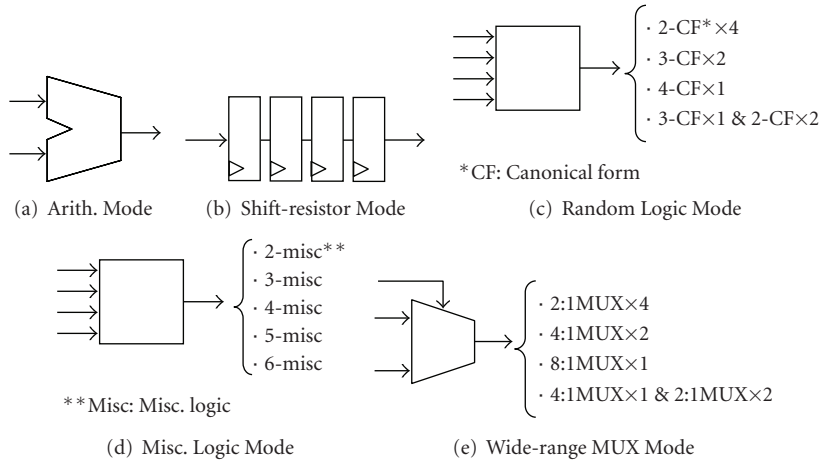


Figure 6: Basic function of a VGLC.

The 2-input Reed-Muller canonical form is described by the following equation:

$$\begin{aligned} F(x_0, x_1) = {} & \{F(0,0)\} \\ & \oplus x_0\{F(0,0) \oplus F(1,0)\} \\ & \oplus x_1\{F(0,0) \oplus F(0,1)\} \\ & \oplus x_0 x_1\{F(0,0) \oplus F(0,1) \oplus F(1,0) \oplus F(1,1)\}. \end{aligned} \tag{1}$$

Each part enclosed by brackets corresponds to a configuration memory bit. This equation can represent the 16 logic patterns with a 4-bit configuration memory as well as a 2-input LUT. Both Figures 3(a) and 3(b) have common parts consisting of EXOR and AND gates. Figure 3(c) shows a hybrid cell (HC) composed of a full adder with four configuration memory bits. The HC can be constructed as either a full adder or a 2-input canonical form according to the computation. In the case of mapping a 1-bit full adder,

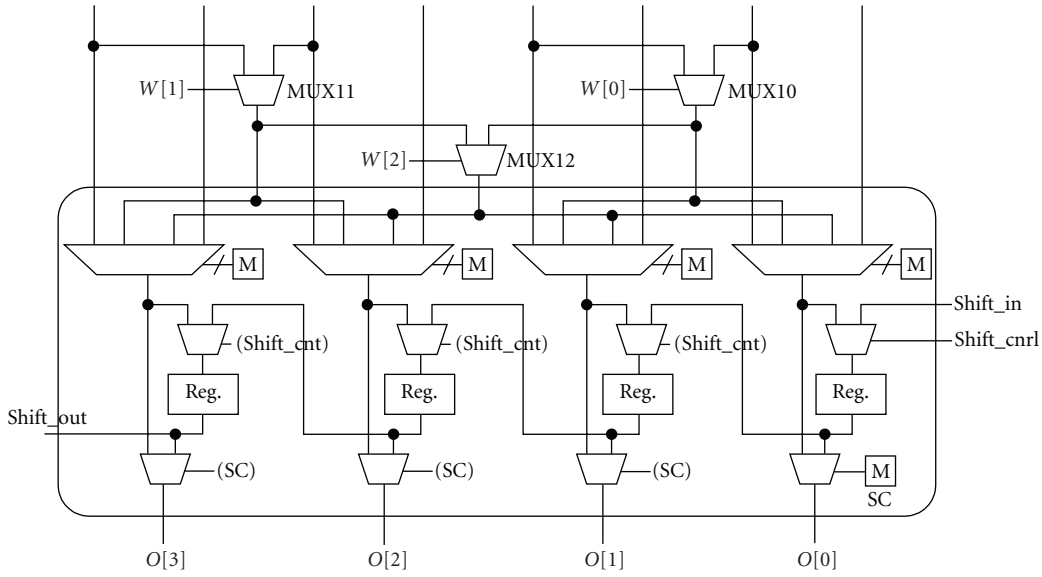


Figure 7: Output selector part.

there is a logic cell structure having either a 4-LUT with carry logic or two 3-LUTs in the traditional FPGAs. In contrast, an HC requires only four configuration memory bits, thereby reducing the configuration data.

Furthermore, to allow increased functionality beyond a 1-bit full adder and 2-input Reed-Muller canonical form, we construct a structure called basic logic element (BLE), as shown in Figure 4. A BLE consists of HCs, MUXes, EXOR, inputs *dc*, *dx*, *dy*, *dz*; and AS, and outputs *dt* and *ds*. Since *dc*, *dx*, *dy*, *dz*, and AS input variables or clamp to 1/0, a BLE can perform three basic functions; as shown in Figure 4.

4.2. Variable Grain Logic Cell Architecture

We propose a VGLC architecture with HC. Figure 5 shows detailed views of the ports in the VGLC. The novel logic block, which has 21 inputs and four outputs with a 27-bit configuration memory, consists of four BLEs and an output selector part. The number of BLEs is determined by two reasons, namely, coarse-grained architecture (e.g., Chess [9] and mixed-grained architecture [5]), which employ nibble bit processing in the logic element and are suitable for extension to 4-, 8-, 16-, 32-, and 64-input ranges. On the other hand, fine-grained architecture uses at least 4-input logic implementation in traditional FPGAs. Therefore, by using a VGLC with four BLEs, the VGLC can implement both nibble bit ALU and 4-input random logic.

The signal carry_in and carry_out terminals are connected directly to the adjoining VGLCs as dedicated lines. The add/sub control signal (AS) and carry selector memory (CP) are entered commonly for all four BLEs. As shown in Figure 6, the VGLC can be programmed to operate in one of five modes: *Arithmetic*, *Shift-resistor*, *Random Logic*, *Misc. Logic*, or *Wide-range MUX Mode*.

4.2.1. Arithmetic Mode

When a BLE is used as a full adder, the VGLC is composed of a 4-bit ripple carry adder or a subtractor with a carry line between the BLEs. Since the carry path is created via the local interconnection within a logic block, it is propagated at a high speed. We can expand the bit range corresponding to the computation using the dedicated lines carry_in and carry_out. It is possible to dynamically select the add/sub functions using AS.

4.2.2. Shift Register Mode

Figure 7 shows the architecture of the output selector part. The output selector with a 9-bit control memory is composed of a serial-to-parallel 4-bit shift register. In addition to the carry path, the VGLC has a dedicated shift line to increase the shift range.

4.2.3. Random Logic Mode

When a BLE is used in the canonical form, the VGLC is composed of a 3-input or 4-input canonical form (3-CF or 4-CF) with MUX10-12 (see Figure 5) using the 2-input canonical form (with some shared inputs). Since 4-LUT is a good implementation in conventional FPGAs, four BLEs are implemented in a single VGLC. In the Random Logic mode, the VGLC can have various canonical forms, for example, four 2-input canonical form, two 3-input canonical form, or one 4-input canonical form, so that suitable BLEs corresponding to the circuit designs can be allocated. Furthermore, the mapping area and the number of configuration memory bits are expected to be improved.

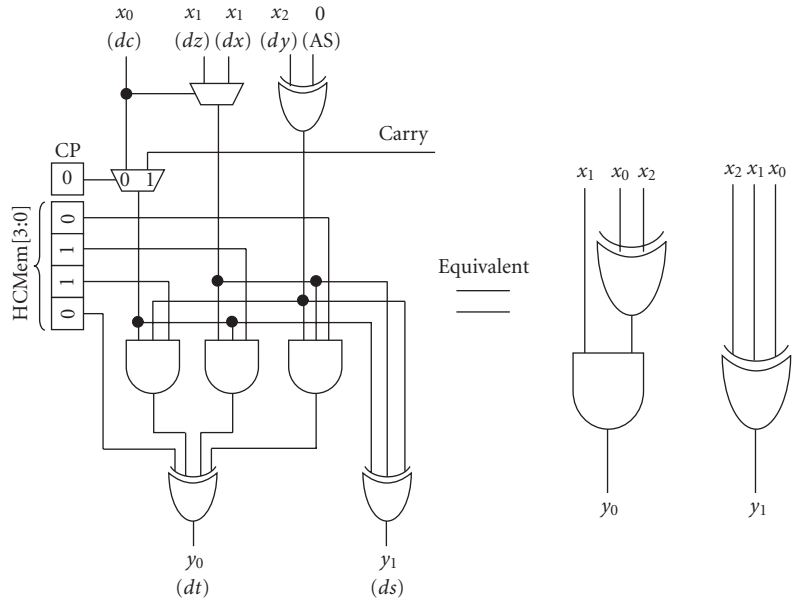


Figure 8: Example of a three-input misc. logic function.

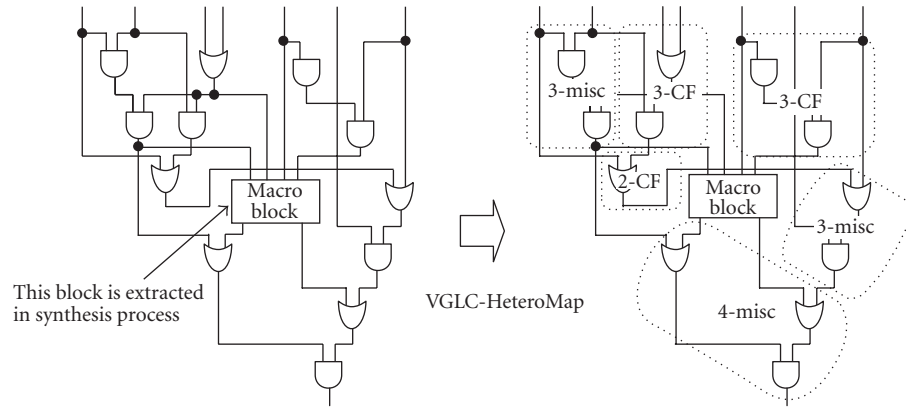


Figure 9: VGLC-HeteroMap.

4.2.4. Misc. Logic Mode

It is shown that the VGLC can represent 2-, 3-, and 4-input canonical forms. However, this requires an area larger than 2-, 3-, and 4-LUTs, respectively. In order to reduce these overheads, we use the misc. logic (miscellaneous logic) function that applies its gate structure. Since a BLE can be used for a 4-input/2-output module, it can represent the maximum 3- or 4-input logic pattern with 4-bit configuration memory bits. Table 2 depicts the logic pattern that can be expressed at the output of dt and ds in the BLE. The K-input variable has 2^{2^K} output logic patterns. Thus, there are 65,536, and 256 patterns in the 4-input and 3-input canonical forms, respectively. For example, since AS = 0 in the 3-input variable, dt and ds are capable of covering 120 and 43 logic patterns, respectively. In total, the misc. logic function has the ability to represent 80.47% of the 3-input logic patterns. Figure 8 shows an example of 3-input Misc. Logic (3-misc.) usage.

Using multiple BLEs, the VGLC can also increase the number of inputs that are expressed. The coverages are 73.6% and 50.3% in the 4-input logic and 5-input logic, respectively, with two BLEs. We can combine a maximum of four BLEs.

4.2.5. Wide Range of Multiplexer Mode

In Figure 5, the output of MUXes 2-5 is through the HC, and the BLE is composed of a 2:1 multiplexer. In addition to the MUXes 10–12, which are used to realize the canonical form, a wide range of MUX configurations, such as 4:1 and 8:1, can be implemented.

```
program VGLC_HeteroMap(network)
  for n := each PI to PO do /* n is node */
    min_label := ∞;
    if n equal macro then
      calculate Label(n) and Cut(n) as macro block;
    else
      for # of input for CF and Misc. Logic do
        calculate Label(n) and Cut(n) as delay and # of inputs;
        if misc. function then
          if pattern is mismatch then continue;
        if Label(n) < min_label then
          min_label := Label(n); min_cut := Cut(n);
        end if
      end for
    Label(n) := min_label; Cut(n) := min_cut;
  end for
  mapping using selected function;
end program
```

Algorithm 1: Pseudocode in random logic mapping.

5. Technology Mapping Method

5.1. Mapping Flow

In order to efficiently map the five functions of the VGLC, the datapath circuits (e.g., adder) a or wide-ranging multiplexer (e.g., 8:1 multiplexer) are prepared in advance in the user library for the logic synthesis process. We can extract these circuits as a macroblock during logic synthesis. The remaining circuits are mapped using the Misc. Logic and Random Logic modes.

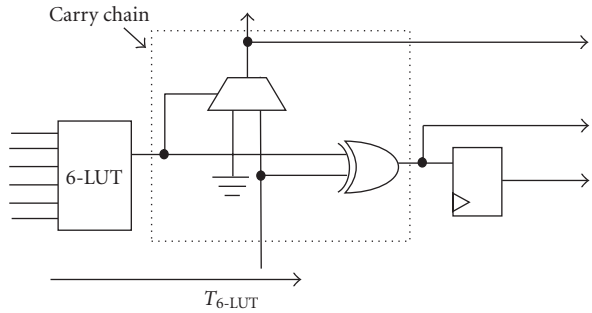
5.2. VGLC-HeteroMap

We need to implement random logic circuits using the VGLC structure. UCLA FlowMap [23] is well known as a FPGA technology mapping algorithm. Although the smallest number in the LUT can be mapped using FlowMap, it targets only homogeneous logic cell structures such as 4-LUT. On the other hand, UCLA HeteroMap [24], which has a delay optimized algorithm, can also map heterogeneous structures, such as the Xilinx XC4K series [25], having a 3-LUT and a 4-LUT in one logic block. Since the HeteroMap source code is open to the public [26], we develop the VGLC-HeteroMap, which is a modified HeteroMap. In particular, we add the following two factors.

(i) Correspondence to the netlist that contains the macroblock.

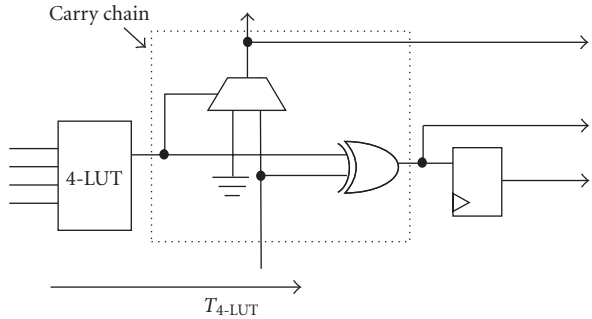
(ii) Addition of the Misc. Logic mode for random logic mapping.

First, since HeteroMap targets single-output LUTs, it must be modified such that the netlist will include a multioutput macroblock. Second, the VGLC has the *Misc. Logic mode* and the *Random Logic mode* for random logic mapping. Therefore, we add the Misc. Logic matching process into the VGLC-HeteroMap. The pattern matching of Misc. Logic results in Boolean matching. Debnath and Sasao [27] present an efficient method to check the equivalence of two Boolean functions under permutation of the variables. As a basis



$T_{6\text{-LUT}} = 0.4\,\text{ns}$
$\text{NB}_{6\text{-LUT}} = 67$
$\text{NT}_{6\text{-LUT}} = 698$

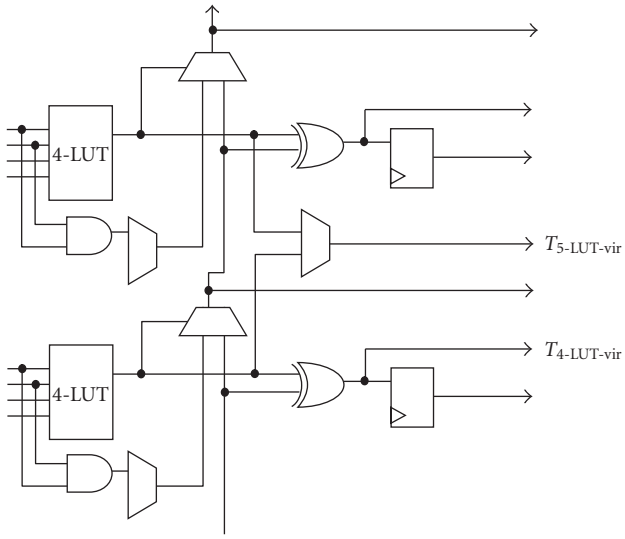
(a) 6-LUT with carry chain



$T_{4\text{-LUT}} = 0.2\,\text{ns}$
$\text{NB}_{4\text{-LUT}} = 16$
$\text{NT}_{4\text{-LUT}} = 196$

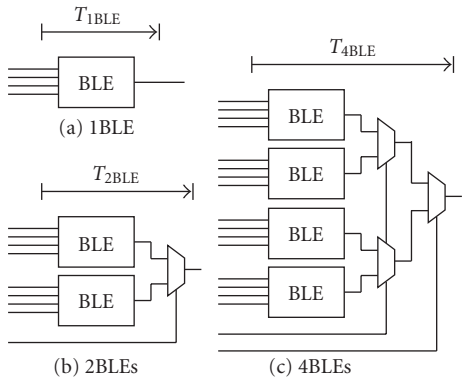
(b) 4-LUT with carry chain

Figure 10: Homogeneous LUT structures with carry chain.



$T_{4\text{-LUT-vir}} = 0.2\,\text{ns}$ $\quad \text{NB}_{\text{Virtex4}} = 35$
$T_{5\text{-LUT-vir}} = 0.46\,\text{ns}$ $\quad \text{NT}_{\text{Virtex4}} = 424$

Figure 11: Heterogeneous LUT structures with carry chain.



$T_{1\text{BLE}} = 2.4\,\text{ns}$ $\quad \text{NB}_{\text{VGLC}} = 27$
$T_{2\text{BLE}} = 2.5\,\text{ns}$ $\quad \text{NT}_{\text{VGLC}} = 666$
$T_{4\text{BLE}} = 2.6\,\text{ns}$

Figure 12: Three components of the VGLC.

of the Boolean matching, they introduced the concept of the P-representative and a breadth-first search algorithm for its quick computation. If two functions have the same P-representative, then they match. We use this technique for Misc. Logic pattern matching. Since this method is useful for up to eight of inputs, we assume that Misc. Logic circuits with a maximum of eight inputs (8-misc.) are used with BLEs. The definitions for this initial search is as follows.

Definition 1. The minterm expansion of an n-variable function is $f(x_1, \ldots, x_n) = c_0 \cdot \overline{x_1 x_2} \cdots \overline{x_n} \vee c_1 \cdot \overline{x_1 x_2} \cdots x_n \vee c_{2^n-1} \cdot x_1 x_2 \cdots x_n$, where $c_0, c_1, \ldots, c_{2^n-1} \in \{0, 1\}$. The binary digit c_j is called the *coefficient of the* jth *minterm*, the jth *coefficient*, or simply the *coefficient*. The 2^n bit binary number $c_0 c_1 \cdots c_{2^n-1}$ is the *binary number representation* of f. The subscript 2 is used to denote a binary number.

Definition 2. Two functions f and g are *P-equivalent* if g can be obtained from f by permutation of the variables. $f \sim^P g$ denotes that f and g are P-equivalent. P-equivalent functions form a *P-equivalence class* of functions.

Definition 3. The function that has the smallest binary number representation among the functions of a P-equivalence class is the *P-representative* of that class.

Figure 9 shows an example of the mapping process in the VGLC. The pseudocode of the VGLC-HeteroMap with the Misc. Logic mode is given in Algorithm 1.

6. Experimental Methodology

In this evaluation, we show the number of logic cells, the logic depth, the area, and the amount of configuration data used to implement benchmark circuits. This section presents a brief description of the environment and the model using the following evaluations.

6.1. Evaluation Environment and Modeling

We need to fairly compare both the coarse-grained and fine-grained types with the VGLC. However, a coarse-grained type logic cell has various structures, and there is no freely available CAD tool. Therefore, the fine-grained logic cell is restricted to three types of LUT-based structure, as shown in Figures 10 and 11. Figure 10(a) shows the 6-LUT, and Figure 10(b) shows a 4-LUT with a carry chain. In addition, Figure 11 shows the heterogeneous LUT structure of Virtex-4. The total number of transistors in each logic cell is computed using the cell count suggested in [28, 29]. A 6-LUT can be implemented using 67 SRAMs, seven inverters, one EXOR, one flip flop, and 64 2:1MUXes, for a total of $67 \times 6 + 7 \times 2 + 1 \times 10 + 1 \times 16 + 64 \times 4 = 698$ transistors. A 4-LUT can be implemented using 16 SRAMs, five inverters, one EXORs, one flip flop, and 16 2:1MUXes, for a total of $16 \times 6 + 5 \times 2 + 1 \times 10 + 1 \times 16 + 16 \times 4 = 196$ transistors. A Virtex-4 structure can be implemented using 35 SRAMs, five inverters, two EXORs, two ANDs, two flip flops, and 35 2:1MUXes, for a total of $35 \times 6 + 5 \times 2 + 2 \times 10 + 2 \times 6 + 2 \times 16 + 35 \times 4 = 424$ transistors. On the other hand, the VGLC has a function based on three types of structure; as shown in Figure 12. The VGLC can be implemented using 27 SRAMs, 12 ANDs, four flip flops, 32 2:1MUXes, and 24 EXORs, for a total of $27 \times 6 + 12 \times 6 + 4 \times 16 + 32 \times 4 + 24 \times 10 = 666$ transistors. Each logic cell is counted as a unit when we evaluate the area. Therefore, the total area of the mapping solution is equal to the total number of logic cells. Such simplification is reasonable because the layout information is not yet available, and the die size of commercial FPGAs is not open.

Similar to the area count, the total amount of configuration data is characterized by the product: the number of configuration memory bits per logic cell $\times$ the total number

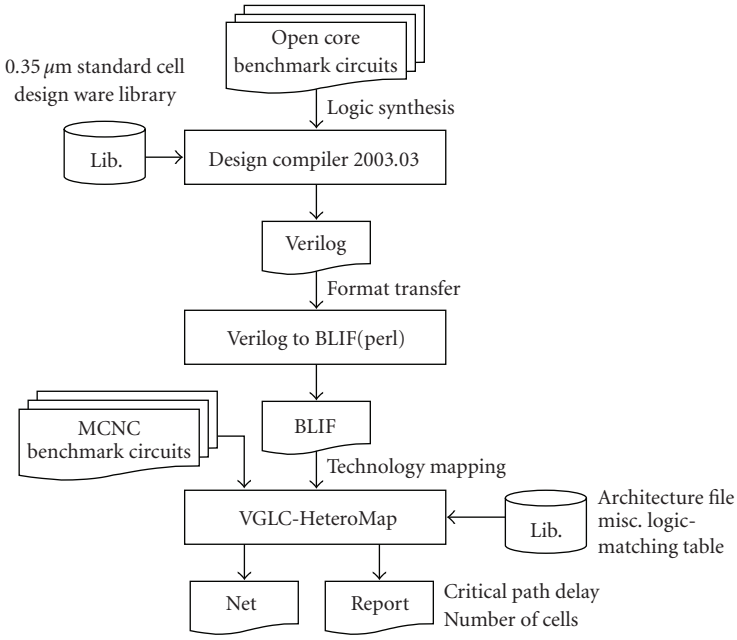


Figure 13: Architecture evaluation flow.

of logic cells. The number of configuration memory bits for each architecture is as follows: VGLC: $NB_{VGLC} = 27$, comparison logic cell: $NB_{6\text{-LUT}} = 67$, $NB_{4\text{-LUT}} = 16$, and $NB_{virtex4} = 35$.

The combination delay of the 4-LUT ($NT_{4\text{-LUT}} = 0.2$ nanoseconds) and heterogeneous LUT ($NT_{4\text{-LUT-vir}} = 0.2$ nanoseconds, $NT_{5\text{-LUT-vir}} = 0.46$ nanoseconds) refers to Xilinx Virtex-4 [21], which is manufactured using 90-nm process technology. In addition, 6-LUT ($NT_{6\text{-LUT}} = 0.4$ nanoseconds) refers to Xilinx Virtex-5 [16], which is manufactured using 65-nm process technology. The delay parameters in the typical condition refer to Xilinx ISE. On the other hand, with regard to transistor optimization for the proposed architecture, we perform transistor level design with the Rohm 0.35-μm 3.3 V transistor model and obtain the delay by Synopsys HSpice simulation in the typical condition. Table 3 summarizes the combination delay for each function of the VGLC.

Note that although the 0.35-μm process technology is somewhat obsolete. In addition, the placement and routing are not performed because the present research focuses on logic depth and area evaluation for the proposed logic cell. The interconnect cost of such an architecture is not considered herein and will instead be evaluated using an advanced process technology in a future study. Therefore, in this evaluation, we assume that the routing architecture is of the island type.

6.2. Evaluation Flow

Figure 13 shows the evaluation flow, and the three mapping methods are as follows:

(A) mapping method using Random Logic mode only,

Table 3: Implementation parameters for VGLC.

Function	$T_{func.}$ [ns]
1-BLE (e.g., 2-CF, 3-misc.)	2.4
2-BLE (e.g., 3-CF, 4-misc., 5-misc.)	2.5
4-BLE (e.g., 4-CF, 6-misc.)	2.6

(B) mapping method using (A) + Misc. Logic mode + macro block,

(C) mapping method using each comparison logic cell.

First, the Verilog RTL netlists of circuits are used as inputs for the synthesis tool Synopsys Design Compiler and are mapped onto the standard cell library. An adder and/or wide-ranging multiplexer is extracted as a macroblock using the DesignWare library. Furthermore, we convert the gate-level netlist into the BLIF format using perl script, the netlists are mapped using the VGLC-HeteroMap. Second, each comparison logic cell used the above-described netlist and technology mapping tool, as well as the VGLC. Finally, we obtain the mapping delay, the logic depth, and the number of logic cells.

In this evaluation, OpenCores [22] circuits and MCNC [30] circuits are chosen as benchmarks. Note that since the MCNC benchmarks are provided in EDIF format, which is almost synthesized, we cannot extract macroblocks.

7. Results and Analysis

7.1. Effect of Misc. Logic

We first evaluate effect of misc. logic functions. In this evaluation, five OpenCores circuits and large/medium MCNC

Table 4: Number of logic cells and mapping delay.

Circuit	No. of logic cells		Mapping delay [ns]	
	(A)	(B)	(A)	(B)
C7552	563	410	19.9	15.4
s5378	430	262	14.9	12.7
C2670	158	96	17.3	12.7
misex3	1,855	1,667	17.7	17.5
seq	1,452	1,006	17.5	17.4
ac97	4,154	2,145	10.1	7.7
aes	11,590	4,633	20.5	19.9
biquad	791	578	25.1	20.2
sha256	3,499	1,719	17.7	14.9
vga	780	385	12.8	12.7

Table 5: Normalized delay, area, and configuration data.

Circuit	Mapping delay		Area		Data	
	(A)	(B)	(A)	(B)	(A)	(B)
C7552	1.00	0.77	1.00	0.73	1.00	0.73
s5378	1.00	0.85	1.00	0.61	1.00	0.61
C2670	1.00	0.73	1.00	0.61	1.00	0.61
misex3	1.00	0.99	1.00	0.90	1.00	0.90
seq	1.00	0.99	1.00	0.69	1.00	0.69
ac97	1.00	0.76	1.00	0.52	1.00	0.52
aes	1.00	0.97	1.00	0.40	1.00	0.40
biquad	1.00	0.80	1.00	0.73	1.00	0.73
sha256	1.00	0.84	1.00	0.49	1.00	0.49
vga	1.00	0.99	1.00	0.49	1.00	0.49
Ave.	1.00	0.87	1.00	0.62	1.00	0.62

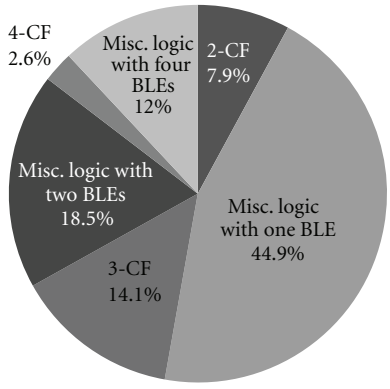


Figure 14: Ratio of Misc. Logic (five MCNC benchmarks).

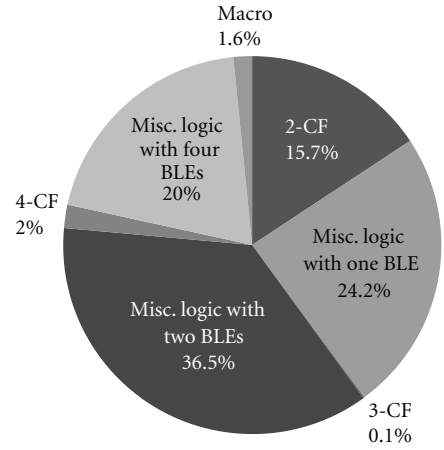


Figure 15: Ratio of Misc. Logic (five OpenCores benchmarks).

benchmarks are used. Table 4 shows the results of the number of logic cells and the mapping delay for technology mapping benchmarks. This table compares two conditions, that is, a VGLC that uses only a canonical form function without the misc. logic function (column method "(A)") and a VGLC that adds the misc. logic function (column method "(B)"). Figures 14 and 15 illustrate the ratio of misc. logic to the total number of logic functions. Note that five OpenCores benchmarks include macroblock functions. misc. logic functions occupy approximately 75.5% and 80.7% in Figures 14 and 15, respectively.

Table 5 shows the mapping delay, the area, and the number of configuration data, which are normalized by each value of (A). Compared to case of CF only, Misc. Logic

Table 6: Logic depth, area, and data for different architectures.

Circuit	VGLC			Virtex-4		
	Depth	Area	Data	Depth	Area	Data
C7552	6	273,060	11,070	8	167,480	13,825
s5378	5	174,492	7,074	6	132,288	10,920
C2670	5	63,936	2,592	7	45,156	3,728
misex3	7	1,110,222	45,009	7	592,328	48,895
seq	7	669,996	27,162	7	506,680	41,825
ac97	3	1,428,570	57,915	4	1,225,996	101,203
aes	8	3,085,578	125,091	8	1,982,624	163,660
biquad	18	384,948	15,606	25	274,328	22,645
sha256	48	1,144,854	46,413	88	1,098,372	90,668
vga	11	256,410	10,395	18	272,632	22,505
circuit	4-LUT			6-LUT		
	Depth	Area	Data	Depth	Area	Data
C7552	8	154,840	12,640	6	316,892	30,418
s5378	6	124,852	10,192	5	291,066	27,939
C2670	7	41,748	3,408	5	76,082	7,303
misex3	7	547,624	44,704	7	1,022,570	98,155
seq	7	468,440	38,240	6	1,173,338	112,627
ac97	4	1,115,828	91,088	4	1,885,298	180,967
aes	8	1,833,776	149,696	7	4,150,308	398,382
biquad	26	255,192	20,832	22	442,532	42,478
sha256	168	1,015,476	82,896	168	2,228,016	213,864
vga	33	250,880	20,480	33	517,916	49,714

Table 7: Normalized logic depth, area, and configuration data.

Circuit	Virtex-4			4-LUT			6-LUT		
	Depth	Area	Data	Depth	Area	Data	Depth	Area	Data
C7552	1.33	0.61	1.25	1.33	0.57	1.14	1.00	1.16	2.75
s5378	1.20	0.76	1.54	1.20	0.72	1.44	1.00	1.67	3.95
C2670	1.40	0.71	1.44	1.40	0.65	1.31	1.00	1.19	2.82
misex3	1.00	0.53	1.09	1.00	0.49	0.99	1.00	0.92	2.18
seq	1.00	0.76	1.54	1.00	0.70	1.41	0.86	1.75	4.15
ac97	1.33	0.86	1.75	1.33	0.78	1.57	1.33	1.32	3.12
aes	1.00	0.64	1.31	1.00	0.59	1.20	0.88	1.35	3.18
biquad	1.39	0.71	1.45	1.44	0.66	1.33	1.22	1.15	2.72
sha256	1.83	0.96	1.95	3.50	0.89	1.79	3.50	1.95	4.61
vga	1.64	1.06	2.16	3.00	0.98	1.97	3.00	2.02	4.78
Min.	1.00	0.53	1.09	1.00	0.49	0.99	0.86	0.92	2.18
Max.	1.83	1.06	2.16	3.50	0.98	1.97	3.50	2.02	4.78
Ave.	1.31	0.76	1.55	1.62	0.70	1.42	1.48	1.45	3.43

improves the critical path delay, the area, and the number of configuration data. The VGLC with misc. logic functions improve one at a maximum of 27% and an average of 13% in mapping delay. Similarly, the area and configuration data improve one at a maximum of 60% and an average of 38%. As a result, Misc. Logic has an effect on the improvement of the circuit performance.

7.2. Comparison with LUT-Based Architectures

Table 6 compares four architectures, that is, the VGLC, the Virtex-4, the 4-LUT, and the 6-LUT using five larger MCNC circuits and five OpenCores circuits (as shown Figure 2). The logic depth, the area, and the number of configuration data are shown in the table. Table 7 shows the performances

normalized by each value of the VGLC. Compared to the Virtex-4 logic cell, the VGLC reduces the logic depth and configuration data by 31% and 55%, respectively, the area increases by 24% on average. Compared to the 4-LUT based logic cell, the VGLC reduces the logic depth and configuration data by 62% and 42%, respectively, while the area increases by 30% on average. Compared to the 6-LUT based logic cell, the VGLC reduces the logic depth, the area, and the configuration data by 48%, 45%, and 243%, respectively. In particular, Sha256 and vga show improvements in logic depth of more than three times the other benchmarks. Since the critical path line contains multibit adder circuits as a macroblock part, such as a 32-bit adder, the VGLC can be effectively packed using the ALU mode. These circuits have a larger arithmetic logic than the other circuits; as shown in Figure 2.

The logic depth and amount of configuration data are reducible in all comparative logic cells. For this reason, the misc. logic function and heterogeneous mapping may have a considerable effect on large-scale applications in particular. With fewer logic cells to cross, the routing delay between the modern FPGA is lower. Therefore, it is necessary to pack more logic into one block in order to avoid routing delay, which is a major problem in the deep submicron FPGA. It is important that the VGLC can pack more logic per logic cells. On the other hand, Table 7 shows that most benchmarks occupy more silicon area, as compared to 4-LUT and Virtex-4 architectures. Nevertheless, the configuration data required by the VGLC is less than that required by all three architectures.

In the present evaluation, we assume that the routing architecture is island style routing. It is actually a routing area which dominates a die area and has a significant influence on circuit performance (area, delay, and power). However, since the VGLC is used as a logic IP core for small or medium circuits, we believe that the number of routing tracks in island style routing is reduced to below the number of stand-alone type modern FPGAs. Moreover, the VGLC does not restrict the routing architecture to the above style, and various connections and routing architectures can be implemented due to the specifications. For example, Runesas MX-core [31] has massive data banks that consist of SRAM data registers, and the medium-grain reconfigurable architecture [32] has a H-tree routing architecture.

8. Conclusions

In the present paper, we have proposed a VGLC architecture and evaluated the area, delay, and configuration data using a technology mapping tool called VGLC-HeteroMap. The novel architecture, which is based on a 4-bit ripple carry adder that includes configuration memory bits, offers a tradeoff between coarse and fine granularity and can be used for efficient mapping in an application. In our evaluation, the VGLC improves the logic depth by 31% and reduces the number of configuration data by 55% on average, compared to the Virtex-4 logic cell.

The present study did not consider the routing network, which is likely to dominate the area and delay in an FPGA implementation. In the future, we will study the connection block and routing structure required to best support the VGLC. Currently, we are attempting to optimize the full custom design to evaluate the chip performance and are developing clustering, place, and routing tools for the proposed architecture. Furthermore, the VGLC can be used as an IP core and can be considered as an FPGA logic element. Since the VGLC can reduce the logic depth compared to other traditional LUT architectures, it is possible to implement the VGLC as a 2D array of the VGLC that is connected by an island style routing architecture. Kuon and Rose [33] showed that the benefit of the IP core in the FPGA could be lost due to its fixed position in the chip, whereas the VGLC has no such restriction. We must also estimate the routing resources using [34] and further explore the use of the VGLC.

Acknowledgments

The present study was supported by the VLSI Design and Education Center (VDEC) at the University of Tokyo in collaboration with Synopsys, Inc., Calif, USA.

References

[1] Y. Satou, M. Amagasaki, H. Miura, et al., "An embedded reconfigurable logic core based on variable grain logic cell architecture," in *Proceedings of International Conference on Field Programmable Technology (ICFPT '07)*, pp. 241–244, Kitakyushu, Japan, December 2007.

[2] R. Yamaguchi, M. Amagasaki, K. Matsuyama, M. Iida, and T. Sueyoshi, "A novel variable grain logic cell architecture with multifunctionality," in *Proceedings of IEEE Region 10 Annual International Technical Conference (TENCON '07)*, pp. 1–4, Taipei, Taiwan, October-November 2007.

[3] M. Amagasaki, R. Yamaguchi, K. Matsuyama, M. Iida, and T. Sueyoshi, "A variable grain logic cell architecture for reconfigurable logic cores," in *Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 550–553, Amsterdam, The Netherlands, August 2007.

[4] M. Amagasaki, T. Shimokawa, K. Matsuyama, et al., "Evaluation of variable grain logic cell architecture for reconfigurable device," in *Proceedings of IFIP International Conference on Very Large Scale Integration (VLSI-SoC '06)*, vol. 2, pp. 198–203, Nice, France, October 2006.

[5] K. Leijten-Nowak and J. L. van Meerbergen, "An FPGA architecture with enhanced datapath functionality," in *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '03)*, pp. 195–204, Monterey, Calif, USA, February 2003.

[6] K. Leijten-Nowak and J. L. van Meerbergen, "Embedded reconfigurable logic core for DSP applications," in *Proceedings of 12th International Conference on Field-Programmable Logic and Applications (FPL '02)*, pp. 89–101, Montpellier, France, September 2002.

[7] S. J. E. Wilton, C. H. Ho, P. H. W. Leong, W. Luk, and B. Quinton, "A synthesizable datapath-oriented embedded FPGA fabric," in *Proceedings of the 15th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '07)*, pp. 33–41, Monterey, Calif, USA, February 2007.

[8] H. Amano, A. Jouraku, and K. Anjo, "A dynamically adaptive hardware on dynamically reconfigurable processor," *IEICE Transactions on Communications*, vol. E86-B, no. 12, pp. 3385–3391, 2003.

[9] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," in *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '99)*, pp. 135–143, Monterey, Calif, USA, February 1999.

[10] T. Sugawara, K. Ide, and T. Sato, "Dynamically reconfigurable processor implemented with IPFlex's DAPDNA technology," *IEICE Transactions on Information and Systems*, vol. E87-D, no. 8, pp. 1997–2003, 2004.

[11] D. Cherepacha and D. Lewis, "DP-FPGA: an FPGA architecture optimized for datapaths," *VLSI Design*, vol. 4, no. 4, pp. 329–343, 1996.

[12] J. G. Delgado-Frias, M. J. Myjak, F. L. Anderson, and D. R. Blum, "A medium-grain reconfigurable cell array for DSP," in *Proceedings of the IASTED International Conference on Circuits, Signals, and Systems (CSS '03)*, pp. 231–236, Cancun, Mexico, May 2003.

[13] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "A novel FPGA logic block for improved arithmetic performance," in *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays (FPGA '08)*, pp. 171–180, Monterey, Calif, USA, February 2008.

[14] Altera, Inc., Altera Stratix II Data Sheet, 2005.

[15] Altera, Inc., Altera Stratix III Data Sheet, 2007.

[16] Xilinx, Inc., Xilinx Virtex-5 Data Sheet, 2007.

[17] M. Iida and T. Sueyoshi, "A shape evaluation of circuit area for reconfigurable logic device," in *Proceedings of International Technical Conference On Circuits/System, Computers and Communications (ITC-CSCC '03)*, vol. 3, pp. 1595–1598, Kang-Won, Korea, July 2003.

[18] T. Higuchi, M. Iwata, D. Keymeule, et al., "Development of adaptive device," in *Proceedings of IEICE General Conference*, vol. A-3-14, p. 100, Kyoto, Japan, March 1998.

[19] P. Jamieson and J. Rose, "Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters," in *Proceedings of IEEE International Conference on Field Programmable Technology (FPT '06)*, pp. 1–8, Bangkok, Thailand, December 2006.

[20] T. Sueyoshi and M. Iida, "Configurable and reconfigurable computing for digital signal processing," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E85-A, no. 3, pp. 591–599, 2002.

[21] Xilinx, Inc., Xilinx Virtex-4 Data Sheet, 2006.

[22] Opencores org, http://www.opencores.org/.

[23] J. Cong and Y. Ding, "Flowmap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 1, pp. 1–12, 1994.

[24] J. Cong and S. Xu, "Delay-oriented technology mapping for heterogeneous FPGAs with bounded resources," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '98)*, pp. 40–45, San Jose, Calif, USA, November 1998.

[25] Xilinx, Inc., Xilinx XC4000 Data Sheet, 1999.

[26] UCLA CAD-LAB, http://cadlab.cs.ucla.edu/.

[27] D. Debnath and T. Sasao, "Fast Boolean matching under permutation by efficient computation of canonical form," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E87-A, no. 12, pp. 3134–3140, 2004.

[28] J. Cong and H. Huang, "Technology mapping and architecture evaluation for k/m-macrocell-based FPGAs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 1, pp. 3–23, 2005.

[29] Y. Hu, S. Das, S. Trimberger, and L. He, "Design, synthesis and evaluation of heterogeneous FPGA with mixed LUTs and macro-gates," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 188–193, San Jose, Calif, USA, November 2007.

[30] K. McElvain, "IWLS'93 benchmark set: version 4.0," in *Proceedings of MCNC International Workshop on Logic Synthesis (IWLS '93)*, Tahoe City, Calif, USA, May 1993.

[31] K. Mizumoto, T. Tanizaki, S. Kobayashi, et al., "A multi matrix-processor core architecture for real-time image processing SoC," in *Proceedings of IEEE Asian Solid-State Circuits Conference (ASSCC '07)*, pp. 180–183, Fukuoka, Japan, November 2007.

[32] M. J. Myjak and J. G. Delgado-Frias, "A medium-grain reconfigurable architecture for DSP: VLSI design, benchmark mapping, and performance," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 1, pp. 14–23, 2008.

[33] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[34] W. M. Fang and J. Rose, "Modeling routing demand for early-stage FPGA architecture development," in *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays (FPGA '08)*, pp. 139–148, Monterey, Calif, USA, February 2008.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 764942, 18 pages
doi:10.1155/2008/764942

Research Article

Architecture-Level Exploration of Alternative Interconnection Schemes Targeting 3D FPGAs: A Software-Supported Methodology

Kostas Siozios, Alexandros Bartzas, and Dimitrios Soudris

Department of Electrical and Computer Engineering, Democritus University of Thrace, 67100 Xanthi, Greece

Correspondence should be addressed to Kostas Siozios, ksiop@ee.duth.gr

Received 6 March 2008; Revised 24 July 2008; Accepted 11 September 2008

Recommended by Lionel Torres

In current reconfigurable architectures, the interconnection structures increasingly contribute more to the delay and power consumption. The demand for increased clock frequencies and logic density (smaller area footprint) makes the problem even more important. Three-dimensional (3D) architectures are able to alleviate this problem by accommodating a number of functional layers, each of which might be fabricated in different technology. However, the benefits of such integration technology have not been sufficiently explored yet. In this paper, we propose a software-supported methodology for exploring and evaluating alternative interconnection schemes for 3D FPGAs. In order to support the proposed methodology, three new CAD tools were developed (part of the 3D MEANDER Design Framework). During our exploration, we study the impact of vertical interconnection between functional layers in a number of design parameters. More specifically, the average gains in operation frequency, power consumption, and wirelength are 35%, 32%, and 13%, respectively, compared to existing 2D FPGAs with identical logic resources. Also, we achieve higher utilization ratio for the vertical interconnections compared to existing approaches by 8% for designing 3D FPGAs, leading to cheaper and more reliable devices.

1. Introduction

In the real-estate market, an often-stated truism is that as land becomes more expensive, there is a tendency to build upwards rather than outwards. This idea has some resonance in the domain of silicon-integrated circuits (ICs), where the size of the die is limited among others by yield and performance constraints. In the next few years, enormous changes are about to happen that will influence the future of LSI. The shift from horizontal to vertical stacking of circuits has the potential to rewrite the conventions of electronics design. Three-dimensional (3D) ICs, which contain multiple functional layers, mitigate many of the limitations introduced by the current process technologies, as they enhance dramatically among others the device performance, the functionality, and the packaging density, as compared to two-dimensional (2D) ones [1].

A qualitative comparison regarding the gains introduced by the 3D integration process, compared to existing system design approaches is summarized in Table 1. More specifically, 3D integration technology provides increased performance in numerous design criteria as compared to the existing 2D approaches, while the wide acceptance of such a fabrication process and the development of supporting CAD tools are still open issues.

Although 3D integration promises considerable benefits, several challenges need to be satisfied. An important challenge is the design space exploration, which is essential to build efficient devices (in terms of high-performance, low energy, electromagnetic interference (EMI), etc.), as well as the design of architectures that exploit all the advantages offered by 3D integration. In addition, CAD tools that facilitate the design of 3D circuits are required. Up to date, there are only a few academic approaches [2, 3] for mapping applications on 3D FPGAs, while there is no complete CAD flow in order to promote the commercialization of this new design paradigm. Furthermore, there is no commercial CAD tool targeting 3D devices, similar to the standalone tools and/or design flows provided by Cadence, Mentor Graphics, and Xilinx for 2D technologies. Consequently, there is

Table 1: Comparison between alternative design implementations.

Property	Single chips	System-on-chip (SoC)	3D integration
Modular flexibility	High	Low	Medium
System performance	Low	Medium-High	High
Physical dimension of products	Large	Medium	Small
Complexity of fabrication process	Low	Medium-High	Medium-High
Fabrication cost	Low	Medium	High
Design methodology, CAD tools	Available	Available	Not deployed yet

an absolute necessity to develop algorithms and software tools to exploit the advantages of the third dimension and solve time-consuming and complex tasks, such as partitioning, placement and routing (P&R) for 3D-reconfigurable architectures.

The benefits of using 3D architectures in logic chips will be especially great for designing field-programmable gate arrays (FPGAs), as these devices always exhibit limitations that occur due to increased wirelength. Compared to ASIC solutions, they consume more power and energy, while they operate in lower frequencies. Since 3D integration technology provides increased number of neighbors, each logic block can access a greater number of nearest neighbors, alleviating the requirement for lengthier connections. Due to this, it is likely that the reconfigurable architectures will drive rapid adoption of 3D IC technology faster than any other device (e.g., ASIC).

Many of the problems alleviated with the usage of 3D integration technology are tightly coupled to the total wirelength. It is common for architecture designers to estimate the longest interconnect equal to twice the length of the die edge. In order to show the potential gains of the new integration approach in this field, Figure 1 illustrates an example structure where the interconnection length is significantly reduced compared to conventional 2D architectures. More specifically, for a given total area equal to A (both for 2D and 3D devices), as the number of layers increases, the area of each layer as well as the longest interconnection is reduced. For instance, if we employ architecture with four layers, the corresponding interconnection length is almost the half (compared to 2D devices).

The 3D integration technology has impact both in physical level (i.e., wirelength, density) as well as product/system level (i.e., performance, power/energy, cost, functionality, and security). In particular, the main design parameters improved by the exploitation of 3D integration are as follows.

(1) *Wirelength*. The interconnection network of large-scale reconfigurable architectures exhibits increased resistance (R) and capacitance (C) values. However, in the 3D approach, the circuits are split up into smaller parts and stacked appropriately alleviating such problems [1, 4, 5].

(2) *Density*. 3D designs support the possibility of implementing more logic in the same footprint area, compared to existing well-established 2D technologies. In other words, the increased functionality extends Moore's law and enables a new generation of tiny but powerful devices.

(3) *Performance*. In current technologies, timing is interconnection-driven. As the propagation delay is proportional to the square of the wirelength, its significant reduction leads to overall performance gains. Furthermore, by shortening the distance, the electrical signals have to travel, 3D interconnect technology could deliver the performance gains promised by Moore's law [6].

(4) *Power/Energy*. Wirelength reduction has an impact on the cycle time and energy dissipation, as the interconnect structures increasingly consume more of the power budgets in modern designs [7].

(5) *Cost*. Three-dimensional reconfigurable architectures can potentially provide a reduction in manufacturing costs, as they might not require the integration of state-of-the-art production lines. The prices of manufacturing equipment have soared with each new generation, and using depreciated equipment they would have a major impact on total cost of development new systems. In addition to this, it is much easier to boost yield with older manufacturing processes, while this cost would drop even further by manufacturing high volume of such 3D FPGAs [6]. Finally, the cost of producing 3D FPGAs is tightly related to the 3D assembly procedure (i.e., die to wafer, wafer to wafer, etc.).

(6) *Functionality*. The design of reconfigurable architectures with three dimensions adds a higher order of connectivity, while it opens a world of new design possibilities/options. By integrating heterogeneous blocks, the derived 3D FPGAs exhibit higher efficiency compared to existing solutions [8]. This feature is even more interesting by allowing components with completely incompatible manufacturing technologies to be combined in the different functional layers of a single 3D device.

(7) *Security*. In addition to that, the 3D integration technology provides more advanced IP security, as the stacked structure makes almost impossible any attempt for reverse engineering. This task can be even more difficult by partitioning the target application in such way so that to obscure the function of each layer.

Recently, many research groups from academia [2–4, 9, 10], industry [11, 12], and research institutes [1] have spent significant effort on designing and manufacturing applications in 3D technologies. Beyne presented a survey of existing 3D fabrication technologies in [1]. This work focuses on available interconnection architectures among the layers of 3D ICs and emphasizes the open issues for current and upcoming 3D technologies. A few companies [11, 12] develop 3D ICs for commercial purposes by stacking wafers,

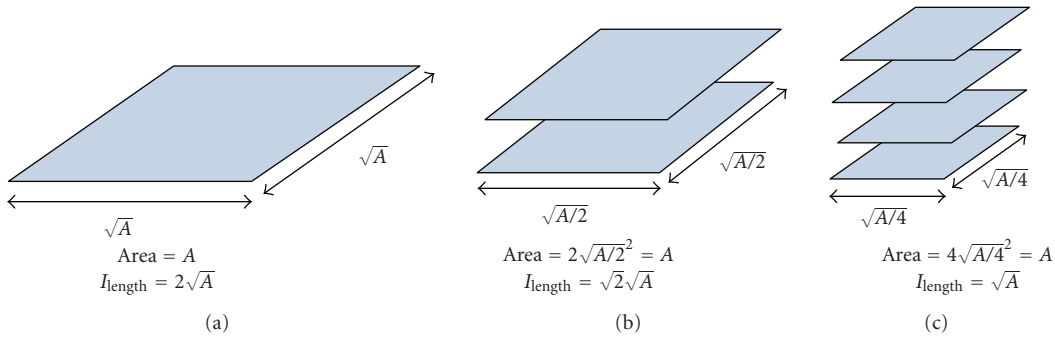


Figure 1: Variation on interconnection length for (a) a 2D device, (b) a 3D architecture with two layers, and (c) a 3D architecture with four layers.

where the distance between the layers is determined by the wafer thickness. Note that the existing industrial research primarily concerns the manufacturing and fabrication processes rather than the development of CAD tools to support the design of emerging 3D technologies.

In [2], the potential of a 3D FPGA technology, based on 3D switch boxes (SBs), is evaluated using analytic models. In order to support the implementation of an application on such a device, a software tool based on [13] is employed. Such an approach exhibits some main drawbacks. First, there is no restriction regarding the amount of vertical connections, which can lead to unacceptable numbers (in terms of fabrication technology) for the 3D device. Moreover, these connections can be formed in any SB, leading to waste of silicon area, while it also increases the fabrication cost of the 3D stack. Furthermore, the assumption that the vertical interconnections are electrical equivalent to routing wires with same length placed on a layer leads to nonrealistic results. Finally, the employed tool cannot estimate/calculate other important design parameters, such as, the energy/power consumption.

An integration process for 3D ICs is presented in [9]. This fabrication technology is based on low-temperature Cu-Cu wafer bonding, where device wafers are bonded in a face-to-back manner with short vertical interconnections. This approach invests effort to minimize either the total wirelength or the number of the inter-layer interconnections. However, other design objectives, such as power consumption or delay are not taken into consideration. Furthermore, the described approach does not permit any architecture level exploration of the target 3D device, as there is no option regarding the hardware resources modification.

A tool flow employed to implement applications on 3D ICs is presented in [3]. The placement algorithm is partitioning-based followed by a simulated-annealing refinement for minimizing the total interconnection length. However, this flow handles only the total wirelength as cost function, ignoring other critical design parameters such as power/energy consumption.

To summarize, in the literature there are two main approaches for designing 3D FPGAs. The first of them affects devices, where each layer can be thought as a "functional layer" [2, 3], while in the second approach each of the layers is specialized (i.e., memory, switches, logic, etc.) [14]. Even though throughout this paper we study a 3D-reconfigurable architecture where all of the layers can be thought to have identical logic resources (we are interested only on the interlayer communication scenario), however, this is not a prerequest, as our proposed methodology can also handle devices with irregular (i.e., heterogeneous) layers.

In this paper, a design methodology for architecture level exploration of alternative interconnection schemes targeting 3D FPGAs is discussed. This methodology is software-supported by three new CAD tools, namely, 3D partitioning (3DPart), 3D placement and routing optimizer (3DPRO), and 3DPower. More specifically, the first one is responsible for the application partitioning to device layers, the second one deals with the placement of each layer and the routing procedure on the 3D-reconfigurable architectures (with full-custom interconnection fabric), while the last one performs the power/energy estimations of these devices. All of them are part of the new Design Framework, named 3D MEANDER [5, 15].

The derived interconnection scheme is integrated into a 3D Virtex-based device for evaluation purposes. During our evaluation procedure, we quantify a number of cost factors. Mainly, we study the application's delay (or performance), the energy consumption, and total wirelength over a plethora of 3D FPGAs with different interconnection schemes. More specifically, the interconnection scenarios affect different number of vertical connections, as well as alternative spatial allocation of them over each functional layer. To the best of our knowledge, the proposed software-supported architecture methodology for exploring/evaluating 3D FPGAs with full-custom interconnections schemes is presented for the first time in the literature. During this evaluation, we prove that we can design 3D architectures with better utilization ratio of vertical interconnection, leading to lower fabrication costs and higher reliability for the 3D devices.

The rest of the paper is organized as follows. In Section 2, we describe the modeling approach of the 3D FPGA architecture, while the proposed architecture exploration methodology and the supporting CAD tools are presented in Sections 3 and 4, respectively. The evaluation results that demonstrate the efficiency of the proposed methodology under numerous design criteria are presented in Section 5,

while the main points of the work are summarized in Section 6.

2. Modeling of the 3D FPGA Architecture

The proposed 3D FPGAs can be constructed by stacking a number of identical 2D functional layers, while we provide the required communication by interlayer through-silicon-vias (TSVs) among vertically adjacent SBs. The architecture of each layer, shown in Figure 2(a), is similar to Xilinx Virtex. In order to model such a device, some of the existing 2D Switches Boxes (SBs) has to be extended to employ connections to the other layers of the 3D FPGA. For our case study, the employed SBs are similar to the ones found in a Xilinx FPGA [16], while more advanced SBs patterns might be found in relevant references. The employed SBs have a permutation function, which defines the track that each routing channel is connected to inside an SB, described by the expression $f(t) = t$, where t is one of the routing tracks [13, 17]. More specifically, this permutation function is shown in Figure 2(b), where the routing tracks of the SB from left side can connect to routing tracks from the rest parts of the SB, marked with the same ID number.

As the implemented permutation function affects the routing efficiency of the target architecture, it also affects the utilization ratio of the 3D TSVs (i.e., 3D SBs). The employed architecture has two flavors of this SB pattern. The first of them affects a 2D SB (as shown in Figure 2(b)), where an incoming routing track can be connected to wires in the three other directions of the SB ($F_s = 3$), whereas the latter (i.e., 3D SB) supports also connections in the third dimension (Figure 2(c)). In the second approach, the incoming routing track is possible to be connected to tracks placed on one of the five other directions (three on the same layer, the upper, and lower layers) ($F_s = 5$). The 2D SB is formed by $6 \times W$ transistors, while the 3D approach requires $15 \times W$ transistors, where W denotes the width of routing channel that crossed in each SB. We have to mention that the interlayer connections occupy much more silicon area, when they are compared to 2D interconnection resources. Due to this, careful selection of the total number of 3D SBs that exist in each of the functional layers, as well as their spatial distribution over the layers, is one of the upmost parameters for steering the optimal selection procedure of the appropriate connectivity across the layers of the 3D device in order to achieve a high-performance and low-power implementation of 3D FPGAs at the minimal fabrication cost

For all of the simulation/evaluation experiments presented in this work, we use a multisegment routing architecture similar to the one that appears in the Xilinx Virtex for the tracks in each layer (composed from routing segments of lengths $L1$, $L2$, $L6$, and long lines, while the distribution of the segments in each channel is 8%, 20%, 60%, and 12%, resp.). An abstract of this multisegment interconnection architecture consisted of wires with lengths $L1$, $L2$, $L6$, and long lines is depicted in Figure 2(d). In order to model the interconnection fabric of each layer, we employ the RC model proposed in [13].

Another critical parameter of the 3D-reconfigurable architecture affects the vertical interconnection that provides the required connectivity among layers. For our study, this communication is realized with through-silicon-vias (TSVs). As this integration technology has not been explored sufficiently yet, careful design of systems that employ such interconnection is required. Also, due to the large variation of the TSV parameters among alternative process technologies, such as diameter, length, dielectric thickness, and fill material, a wide range of measured resistances, capacitances, and inductances have been reported in the literature [11, 12, 18–21].

Electrical characterization of these structures is a crucial requirement since electrical models are necessary to accurately describe the interconnect power and speed of a 3D circuit. The employed values of electrical equivalent circuit, shown in (1), for each TSV, are based on existing approach from [21]:

$$L_{\text{TSV}} = \frac{L_0}{1 + \log\left(f/10^8\right)^{0.26}}, \qquad R_{\text{TSV}} = R_0 \times \sqrt{1 + \frac{f}{10^8}}. \tag{1}$$

In order to model the impact of TSVs on the 3D FPGA (i.e., ground-signal-ground TSV configuration), we employ a high-frequency equivalent circuit, shown in Figure 3. More specifically, the electrical model of each TSV is expressed as a resistor (R_{TSV}) and an inductor (L_{TSV}), while the capacitive coupling between the TSVs is modeled as coupling capacitors (C_{ox}, C_{si} and C_{TSV}). Regarding the parameter C_{TSV}, it denotes the capacitance of the thin oxide layer surrounding the TSV barrel, while the C_{ox} corresponds to the capacitance of the oxide layer on the silicon surface and the fringing field between the TSVs. The capacitance of the silicon substrate is denoted by C_{si}, and the loss property of the silicon substrate between the signal TSV and the ground TSV is denoted by G_{si}. For this setup, the values of these parameters are $C_{\text{TSV}} = 910\,\text{fF}$, $G_{\text{si}} = 1.69\,\text{m}/\Omega$, $C_{\text{si}} = 9\,\text{fF}$, and $C_{\text{ox}} = 3\,\text{fF}$. In these equations L_0 and R_0 correspond to the idunctance and the resistance of a TSV, respectively. For a frequency of 0.1 GHz, considering skin effect of the via barrel, the values of these parameters are $L_0 = 35\,\text{pH}$ and $R_0 = 12\,\text{m}\Omega$, respectively. Based on exploration results shown in [21], it is proven that such a TSV modeling exhibits negligible error for operation frequencies up to 20 GHz.

3. Exploration Methodology for Building 3D FPGAs

The proposed methodology for exploring alternative interconnection schemes for 3D-reconfigurable architectures is composed by three steps, as they are depicted in Figure 4, each of which is implemented as a CAD tool.

Figure 5 shows a detailed description of each methodology step. The input to this methodology is the application graph that describes the functionality of the digital system. The first step of the methodology deals with the application's

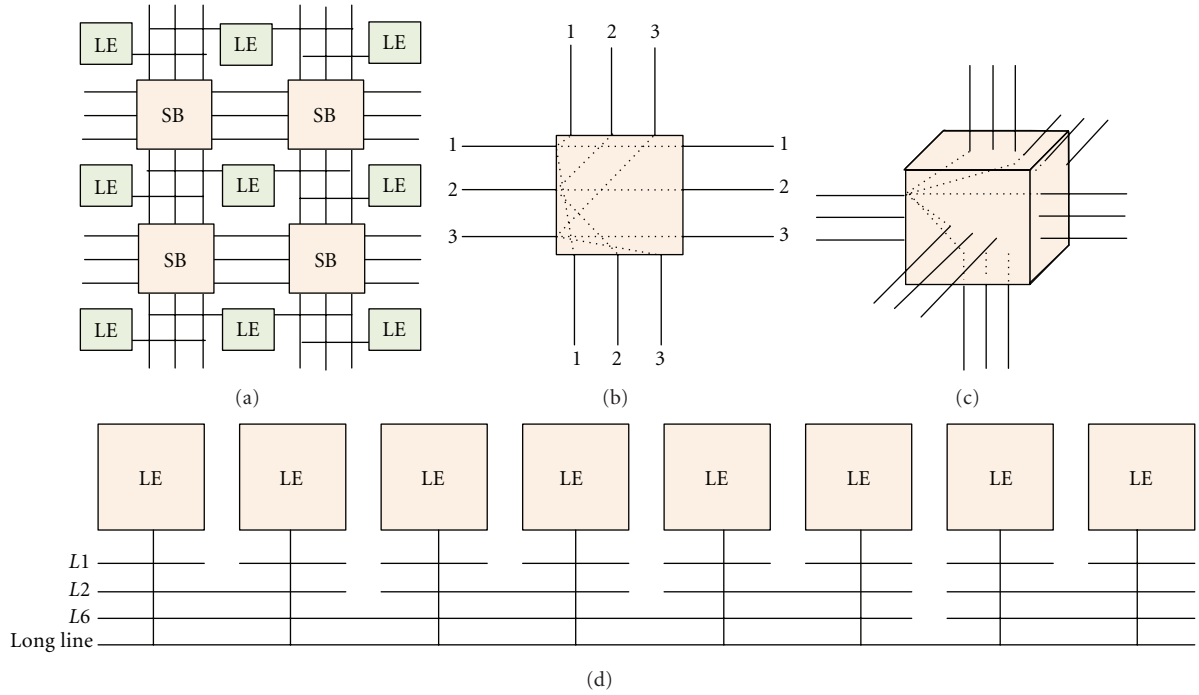


Figure 2: An abstract view of different parts from our architecture: (a) part from the device layer, (b) a 2D SB, (c) a 3D SB, and (d) a multisegment interconnection architecture.

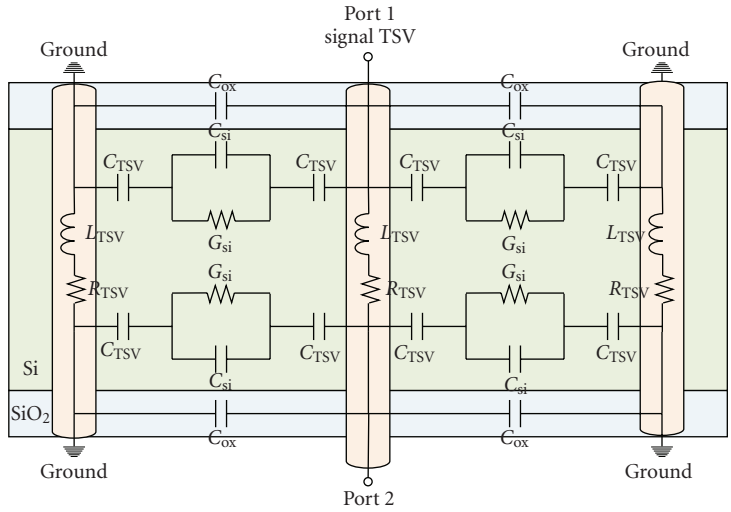


Figure 3: The electrical equivalent circuit for modeling a TSV [21].

graph partitioning, assignment to 3D device layers, and layer ordering. The procedure of splitting an application to a number of parts, which essentially assigns these parts to the available functional layers and orders them to build the 3D FPGA with reasonable execution times, is implemented in the *3DPart* tool by incorporating Pareto-based methods [22]. Such an approach utilizes in a better way the available hardware resources of each layer.

In contrast to existing solutions for application partitioning on 2D [23] or 3D FPGAs [2, 3], which mainly focus on a *min-cut* approach [23], our partitioning algorithm exhibits higher flexibility (as it takes into consideration additional constraints, such as area balance or power/temperature distribution), leading to more accurate partitions. The employed cost functions, which steer the algorithms of the partitioning step of our proposed methodology, provide a tradeoff between the required number of TSVs and the application metrics (such as delay, power/energy consumption). We have to mention that in contrast to a conventional min-cut approach, our proposed algorithm is aware about the number of interlayer connections between successive layers, while it also pays effort to balance

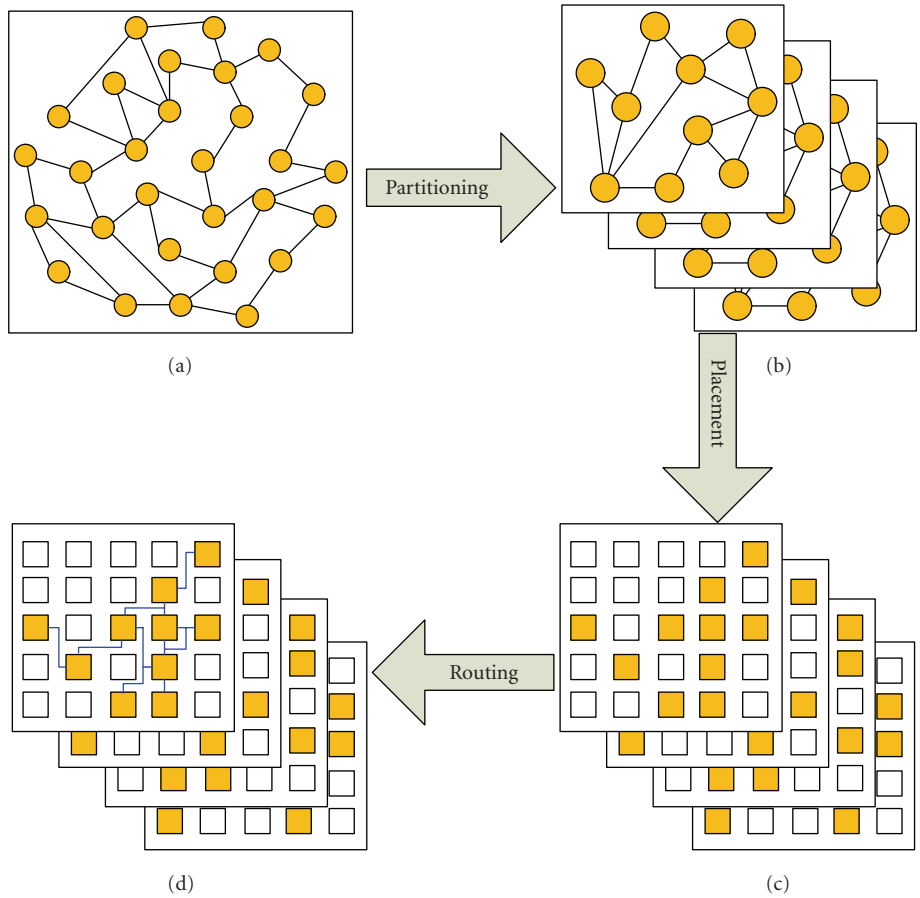


Figure 4: The design procedure for application mapping on 3D FPGAs: (a) initial application graph, (b) application partitioned to layers, (c) application placement, and (d) application routing to target 3D device.

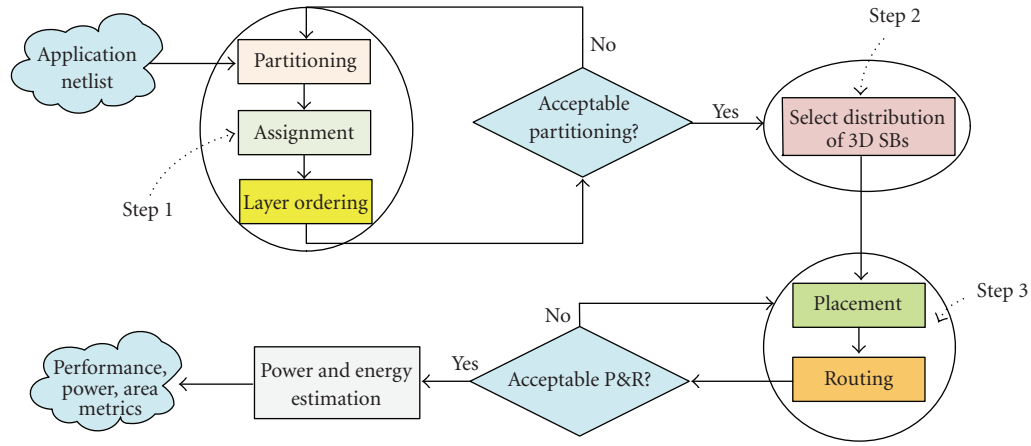


Figure 5: The proposed methodology for exploring alternative 3D-reconfigurable architectures.

them over the 3D device. Furthermore, the modularity of the exploration framework gives the opportunity to the designer to develop and employ more advanced cost functions.

By the end of the first step, we have assigned the logic functionality of the application to the available hardware resources placed on each layer of the 3D FPGA. In order to evaluate the derived result, we quantify the efficiency of the partitioning, in terms of numerous design parameters. More specifically, the resulted decision about the derived partitioning is based on the interlayer connectivity demand (i.e., number of required TSVs), the estimation

about power/delay, as well as the power distribution among layers. Based on the design goals for the derived 3D-reconfigurable architecture, the output of the first step is either accepted or not. Whether an acceptable solution is derived, we proceed to the second step of our methodology. On the other hand (when the partition does not meet the design constraints), the application is fed back to the first step. Employing the 3DPart tool performs this step.

The second step of the proposed methodology deals with the selection and distribution of 3D SBs. As we design FPGAs, it is well worth to distribute them uniformly over the layer's area. Even though, more advanced distributions might be found in relevant approaches [5], they lead to increase design cost due to the higher complexity. Consequently, for our study, we select to distribute the available number of 3D SBs, uniformly over the layer's area.

The third step of the proposed methodology deals with the application's placement and routing (P&R) on a 3D-reconfigurable architecture, while it is software supported by the 3DPRO tool. During this step, the logic functions of each layer are assigned to hardware blocks placed on specific spatial locations (x_i, y_i, z_i), while the appropriate interconnections among them are formed by the routing resources. Different cost functions might be employed during the P&R steps. More specifically, based on the application constraints, it is possible to use either a connectivity-aware or a power-aware approach. In the first case, the logic functions are placed and routed by having as goal to achieve as high as possible operation frequencies, while in the latter approach alternative design parameters (such as power dissipation) are the primary design goals.

By the end of this step, we have the complete P&R of the application on the 3D FPGA device. In order to prove the effectiveness of the proposed methodology, we study the maximum operation frequency, the power/energy consumption, as well as the hardware resources utilization (in terms of routing fabric). The last parameter is very crucial to determine the percentage of utilized vertical interconnections (i.e., TSVs), since they exhibit increased fabrication cost. When the P&R does not meet the designer's criteria, there is a feedback to the third step of the proposed methodology for additional improvements.

In order to determine the power/energy consumption of the application implemented onto the derived 3D-reconfigurable architecture, we use a new CAD tool, named 3DPower. This tool incorporates existing power models proposed in [24]. These models were extended to handle sufficiently the extra hardware parameters (such as multiple layers, TSV connections, etc.) introduced by the integration on the third dimension.

4. Meander Framework for 3D FPGAs

The proposed exploration methodology for building sufficient 3D-reconfigurable architectures is software supported by three new CAD tools, named 3DPart, 3DPRO, and 3DPower. These tools are part from the 3D MEANDER design framework [5, 15, 16], depicted in Figure 6. This flow utilizes existing CAD tools from the 2D toolset, which do not need to be aware of the three-dimensional FPGA topology. More specifically, new tools replaced the P&R and power consumption estimation, as these tasks consider the particular features of the 3D FPGAs. We have replaced the current version of P&R tool of the 2D flow (i.e., EX-VPR [25]) with the proposed P&R tool, named 3DPRO. We have also replaced the existing PowerModel tool, with the new 3DPower for modeling and calculating power/energy consumption in 3D architectures, while we have added an additional tool, named 3DPart, which deals with the application partitioning to 3D stack. To the best of our knowledge, this toolset is the first complete framework in academia for exploring alternative 3D-reconfigurable architectures starting from a hardware description language (HDL) and ending up to configuration file generation. Next, we describe in more detail the employed algorithms, as well as their software implementation, regarding the three new CAD tools.

4.1. Partitioning

The first of the new CAD tool deals with three tasks: (i) the application's partitioning, (ii) the partitioning to layer assignment, and (iii) the layer ordering. All of them are crucial for efficient implementation onto 3D architectures. Up to now, many years of research have been spent to develop fast and accurate algorithms just for supporting the first task (i.e., the application's partitioning). As the three-dimensional integration technologies are not studied efficiently yet, we are not aware about any other existing tool either for assigning partitions to devices layers of a 3D FPGA or order these layers.

An efficient partitioning algorithm can alleviate a number of design problems. Among others, by assigning closely layers that exhibit high data transfers, it is feasible to achieve higher operation frequencies. In addition to that, clusters of functions with high bandwidth requirements should be assigned to logic blocks that belong to the same layer, as there is plethora of routing resources compared to the reduced resources available for interlayer connectivity. Moreover, the appropriate selection of layer ordering, based on the existing power sources on them, might prevent failures related to heat dissipation. Finally, the layer ordering might alleviate congestion problems, as the interlayer communication resources are limited compared to routing wires of each layer.

The development of research in partitioning in the past two decades can be found in a comprehensive survey [26]. As the performance variation regarding numerous design parameters is tightly firmed to the employed interlayer communication fabric, a good partitioning among others have to limit the number of signals travelling through layers. This constraint is also known as a min-cut partitioning approach. However, apart from the min-cut, which is thought to be the objective for relevant approaches [2, 3, 23], the proposed one also takes into consideration additional design parameters (i.e., spatial distribution of

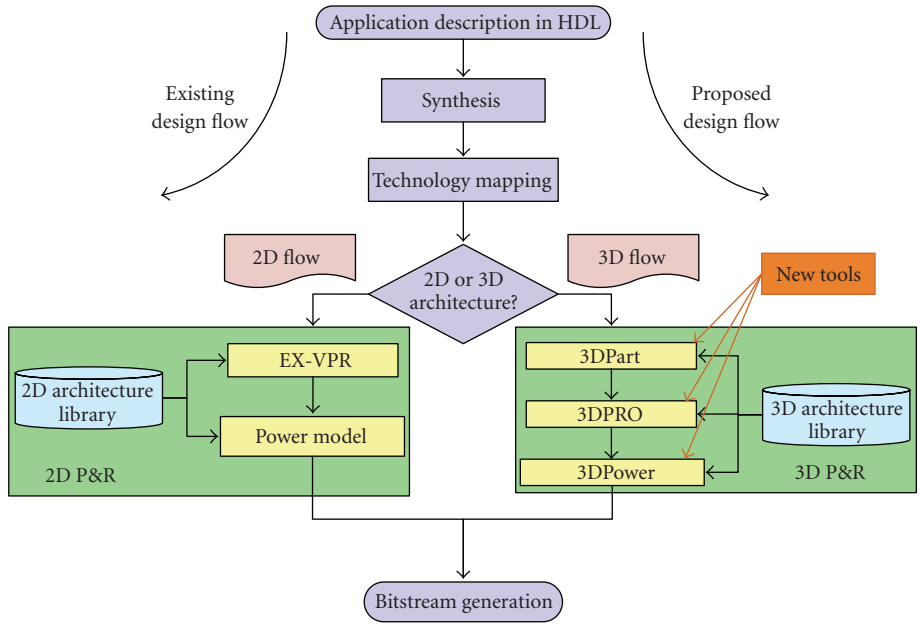


Figure 6: The MEANDER framework: (a) the left branch concerns the design of 2D conventional FPGAs and (b) the right branch concerns the design of 3D FPGAs.

TSVs among layers, area per layer, operation frequency, etc.).

The proposed algorithm that realizes these tasks is shown in Algorithm 1. Initially, the algorithm calculates the partitioning of application's hypergraph (i.e., application netlist) to a number (equals at least to the total device layers). Then, two iterative loops are applied in order to find the layer to which each of the logic functions has to be assigned. More specifically, firstly we are interest to derive an acceptable assignment of partitions to the device layers (in terms of the design constraints), then we try to determine the optimal ordering of these layers. Whether the number of partitions is higher compared to the device layers, then during the first task, more than one partition might be combined and assigned to the same layer of the 3D FPGA.

4.2. Placement Algorithm

After splitting the application to device layers, the placement algorithm assigns the application's ith logic function to the available hardware logic block, placed on physical location (x_i, y_i, z_i). The placement algorithm is based on simulated annealing. By analogy with this physical process, each step of the simulated annealing algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter called temperature, which is gradually decreased during the process. More specifically, during the execution of the placement algorithm, pairs of logic blocks are selected and swapped randomly, until either the resulted placement is good enough or the maximum number of iterations is reached. The efficiency of a placement is characterized by calculating the cost function, shown in (2) and (3):

$$\Delta\text{Cost} = \alpha \times \frac{\Delta\text{Wire\_cost}}{\text{Previous Wire\_cost}} + (1-\alpha)\frac{\Delta\text{Time}_{\text{cost}}}{\text{Previous Time}_{\text{cost}}}, \tag{2}$$

where

$$\text{Timing}_{\text{cost}} = \sum_{\forall i,j \in \text{application}} \{\text{Delay}(i,j) \times \text{criticality}(i,j)^{\text{const}}\},$$

$$\text{Wiring\_cost} = \sum_{i=1}^{\text{Total\_Nets}} \left\{ q(i) \times \left[\left(\frac{\text{bb}_x(i)}{C^{\beta}_{\text{av},x}(i)} + \frac{\text{bb}_y(i)}{C^{\beta}_{\text{av},y}(i)} \right) + \left(\varepsilon \times \frac{\text{bb}_z(i)}{C^{\gamma}_{\text{av},z}(i)} \right) \right] \right\}. \tag{3}$$

Whenever the value of this cost function is reduced, the swap is kept. However, if the cost value increases, then the probability of keeping the swap is reduced with the execution time.

In this cost function, the factor α balances the effort of placement algorithm to optimize either the wirelength or the application's delay. The delay(i, j) denotes the delay between the logic elements i and j (a source-sink path of a network), the factor *const* is a constant, while the criticallity(i, j) gives the importance in terms of how close to the critical path is the network i. Similar to [13, 25], its mathematic expression is defined as Criticallity(i, j) = $1 - \text{Slack}(i, j)/\text{Delay}_{\text{max}}$, where $\text{delay}_{\text{max}}$ is the delay of the circuit critical path, and slack(i, j) is the amount of delay

```
while (accept partition ← True) do
  {
   subgraphs ← split (netlist, number of layers);
   while (accept partition to layer assignment ← True) do
     {
      while (accept layer ordering ← True) do
        {
         connections ← calculate (interconnections among subgraphs);
         estimate variation among partitions (power, area, delay);
         goal ← evaluate retrieved partitioning;
         if (goal not optimal) then
           {
            try to repartition the netlist ();
           }
         else
           {
            accept partition ← True;
           }
        }
     }
  }
```

Algorithm 1: Algorithm for application partitioning, partitions to layer assignment and layer ordering tasks.

```
P ← Initial_Random_{Placement}();
T ← Initial_Temperature();
R_limit ← Initial_R_limit ();
while (Exit_Criterion() not TRUE) // outer loop
  {
   while (loop_criterion() not TRUE) // inner loop
     {
      P_new ← Random_swap_Placements(P, R_limit);
      ΔCost ← Cost(P_new) − Cost(P);
      r ← Random_value(0, 1);
      if (r < e^(−ΔCT)) // accept the movement
        {
         P ← P_new;
        }
     }
   R_limit ← Update (R_limit);
   T ← Update (Temperature);
  }
```

Algorithm 2: The proposed simulating annealing-based algorithms for placement on 3D FPGAs.

that could be added to this connection before it affected the application's critical path. The factors $\mathrm{bb}_x(i)$, $\mathrm{bb}_y(i)$, and $\mathrm{bb}_z(i)$ denote the dimensions of the 3D bounding box for network i, while the $q(i)$ is a scaling factor for this bounding box, used to make more accurate estimations about the wire-length for nets with more than 3 terminals [13]. The $C_{\mathrm{av},x}(i)$, $C_{\mathrm{av},y}(i)$, and $C_{\mathrm{av},z}(i)$ parameters correspond to the average width of routing tracks on x, y, and z axis of the bounding box for network i, while they are used in order to force placement algorithm to take into consideration the available (fabricated) routing resources. The value of these parameters depends solely on the fabricated interconnection resources, while it is constant during the placement. The values of β and γ control the relative cost of employing narrower and wider routing channels. More specifically, when their values are 0, then the cost function results in the conventional bounding box approach. Otherwise, as higher the values of these parameters are, then more and more tracks from narrowest routing channels have increased cost value, compared to the wider channels. We employ a different relative cost (γ) for the TSVs, as the placement algorithm has to pay effort to not waste this kind of connections. Finally, by using an additional factor, denoted as ε, we discourage the placer to put functions in different layers.

Algorithm 2 shows the proposed 3D placement algorithm which is realized as part of the 3DPRO CAD tool. As it was already mentioned, the functionality of this approach is based on simulated annealing. In order to obtain high-quality solutions in a reasonable computation time with such an approach, a good annealing schedule is essential. The proposed schedule incorporates some of the features provided in relevant references [13, 25, 27–29], while we propose a new temperature update scheme, as well as an exit criterion. The total moves per temperature are equal to $N = m \times (\mathrm{LE})^{4/3}$, where m denotes a default number of moves (usually 10), while the LE represents the number of logic elements that build the FPGA. This approach is similar to the one found in existing approaches [13, 25, 28].

In this algorithm, the value of P denotes each of the derived placements; R_{limit} determines the maximum horizontal and vertical distance between two logic blocks that are swapped during the annealing procedure, while its value is reduced linearly during the algorithm's execution.

For our specific architecture, the mathematic expression that describes this reduction is shown in (4):

$$R_{\text{limit}} = d \times \text{Previous\_}R_{\text{limit}}. \tag{4}$$

Whenever the value of R_{limit} is small enough, then the swaps of logic blocks occur for relative closely placed blocks. Such local swaps tend to result in relatively small changes in the placement cost, increasing their probability of acceptance. Initially, the value of this parameter is set to span of the entire layer, while whenever the temperature is updated, then this value is recalculated.

The temperature update is defined by (5):

$$T_i = T_0 \times e^{-Ai}, \quad \text{where } A = \frac{1}{N} \times \ln\left(\frac{T_0}{T_N}\right). \tag{5}$$

In this equation, T_i is the temperature for iteration i, where i increases from 0 to N. Regarding the T_0 and T_N parameters, they correspond to initial and final temperatures, respectively. The employed temperature update scheme guarantees that we will result very closely to the optimal placement. Employing an annealing procedure that spends more time at the most productive temperatures (those that a significant fraction of moves is being accepted), compared to the case where temperature is high (almost any swap is kept), leads to significant improvement in placement's cost. We have to mention that, in practice, the ideal cooling rate cannot be determined beforehand, and should be empirically adjusted for each problem. The procedure of swapping the spatial location of logic blocks (i.e., annealing) is continued as far as the temperature is higher than a small fraction of the average cost of a net. After that point, any movement that results in increase of cost is unlikely to be accepted.

4.3. Routing Algorithm

By defining the placement of logic functions on the 3D FPGA, the routing algorithm forms the appropriate connections among the utilized hardware blocks through the available interconnection fabric. The proposed routing algorithm is an extended version of the Pathfinder negotiated congestion [30]. During the first iterations, a number of networks are allowed to share the same routing fabric. However, as the number of iterations increases, this is gradually prohibited, until the final routing, where each network uses dedicated routing fabric. The proposed routing algorithm can find the narrowest horizontal and vertical channel widths for which the application is fully routable.

As the vertical interconnections among layers are limited, the routing algorithm sets the weights of TSVs to a higher value (compared to routing wires of each plane) in order to discourage the router to form unnecessary bends between horizontal and vertical wires. Also, this forces the router not to connect logic blocks placed on one layer by using interconnection fabric from different layers.

The employed cost function that guides the proposed routing algorithm follows:

$$\begin{aligned}\Delta\text{Cost}(n) = {} & [\text{Criticallity}(i,j) \times \text{Delay}(n)] \\ & + (1 - \text{Criticallity}(i,j)) \times [b(n) \times h(n) \times p(n)],\end{aligned} \tag{6}$$

where Delay(n) is the delay of hardware component n, while the parameters $b(n)$, $h(n)$, and $p(n)$ represent the base cost, the historical congestion cost, and the present congestion cost for the hardware component n, respectively. In order to come to acceptable solutions without overusing the routing resources, the value of $p(n)$ increases with the execution time.

4.4. Power Estimation

The third developed tool, named *3DPower*, is responsible for the modeling and calculation of energy/power consumption for applications implemented onto 3D FPGAs. This tool adopts some principles from existing work proposed in [24] regarding conventional (2D) reconfigurable architectures. However, its models are refined extensively in order to be aware about a number of (heterogeneous) functional layers, as well as the multiple fabrication technologies of 3D stacked ICs. The pseudocode of this algorithm is shown in Algorithm 3.

In this algorithm, the transition density is an efficient measure of the switching activity of each signal within the circuit. Such a model has two parameters [24]:

(1) transition density (7) that denotes the average number of transitions per unit time, where $n_x(T)$ represents the number of transitions within time T:

$$D(x) = \lim_{T \to \infty} \frac{n_x(T)}{T}, \tag{7}$$

(2) static probability (8) that corresponds to the probability of the signal being high for a certain time period:

$$P(x) = \lim_{T \to \infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t)dt. \tag{8}$$

5. Exploration and Comparison Results

This section provides both qualitative and quantitative comparisons among the proposed methodology for implementing digital applications on 3D-reconfigurable architectures, compared to alternative solutions that can be found in relevant literature. Since the efficiency of application implementation on 3D FPGAs depends mainly on the employed P&R algorithms, we perform a qualitative comparison among our proposed tool (3DPRO), the PR3D [3], and the TPR [2], which are the only available tools for P&R on 3D FPGAs. The results are summarized in Table 2. Given a 3D topology, the proposed methodology, and hence the CAD tool, can explore a plethora of parameters such as delay, energy/power

```
for i = 0 to Total Networks
    {
    for each Logic Block that form Network i
        {
        calculate static probability();
        calculate transition density; ()
        }
    calculate activity of net i();
    net_power = 0;
    for each segment used to route this net
        {
        calculate capacitance of segment();
        net power = net power + switching power for this network();
        }
    total power = total power + net power;
    write power file;
```

Algorithm 3: Algorithm of the *3DPower* tool for 3D FPGAs.

Table 2: Qualitative comparison between TPR and our proposed solution.

Feature	TPR [2]	PR3D [3]	3DPRO (Proposed)
Architecture exploration	Yes	No	Yes
Measure delay	Yes	Yes	Yes
Measure wirelength	Yes	Yes	Yes
Measure power	No	Yes	Yes
Supported switch boxes	Subset Wilton Universal	ASIC devices	Designer specified
Heterogeneous interconnect (simultaneously 2D/3D SBs)	No	Yes	Yes
Vias exploration	No	No	Yes
Part of complete framework	No	No	Yes

consumption, leakage power, and silicon area. Furthermore, it supports the evaluation of alternative architectures, in terms of fabricated TSVs, while the TPR [2] employs a full-connectivity scenario, where each SB can form connections on the adjacent stacked functional layers. However, this scenario does not correspond to a realistic approach for the 3D P&R problem, as the TSVs are prefabricated before the design implementation. Also, it is not possible to integrate so high number (or density) of TSVs per layer, as they occupy significant area, increasing among others the yield cost. Consequently, the 3DPRO provides more flexibility to the designer in order to perform architecture level exploration.

The effectiveness of the proposed methodology is evaluated with the usage of the 20 largest benchmarks from the Microelectronics Center of North Carolina (MCNC) benchmark suite [31]. For each of them, we perform an exploration regarding various design parameters.

During our exploration methodology we study the impact of alternative distribution of 3D SBs into 3D FPGAs. More specifically, we quantify the potential gains of employing the available interconnections schemes shown in Figure 7, each of which has advantages and disadvantages. Among others, the approach, where all the SBs of the layer form connections to the rest layers (shown in Figure 7(a)), provides the maximum connectivity improvement, in a penalty of the increased silicon area occupied from so high amount of TSVs. The second and third approaches affect distribution scenarios, where the 3D SBs are assigned either to the center (Figure 7(b)) or the periphery (Figure 7(c)) of each layer, respectively. Both of these implementations exhibit a piece-wise regular interconnection architecture, which leads to retrieve potential gains from the employed supporting CAD tools (i.e., P&R). On the other hand, such approaches might increase either the wirelength or the spatial distribution of other crucial design parameters (i.e., distribution of power sources). For instance, regarding Figure 7(b), the center of the layer exhibits significant higher connectivity demands, which might result to increased power dissipation, or on-chip temperature values. Finally, the last approach (shown in Figure 7(d)) affects a full-custom assignment for 3D SBs, where these connections are assigned based on the connectivity demands for interlayer connectivity, introduced either from a specific application (i.e., MPEG4, GSM) or from an application domain (i.e., multimedia, communications, etc).

Based on the previous conclusions, we might derive that none of these assignments of 3D SBs is efficient in terms of performance, power consumption, and silicon area. Due to this, throughout this paper, we employ a new scheme for assigning 3D SBs, which is depicted in Figure 8.

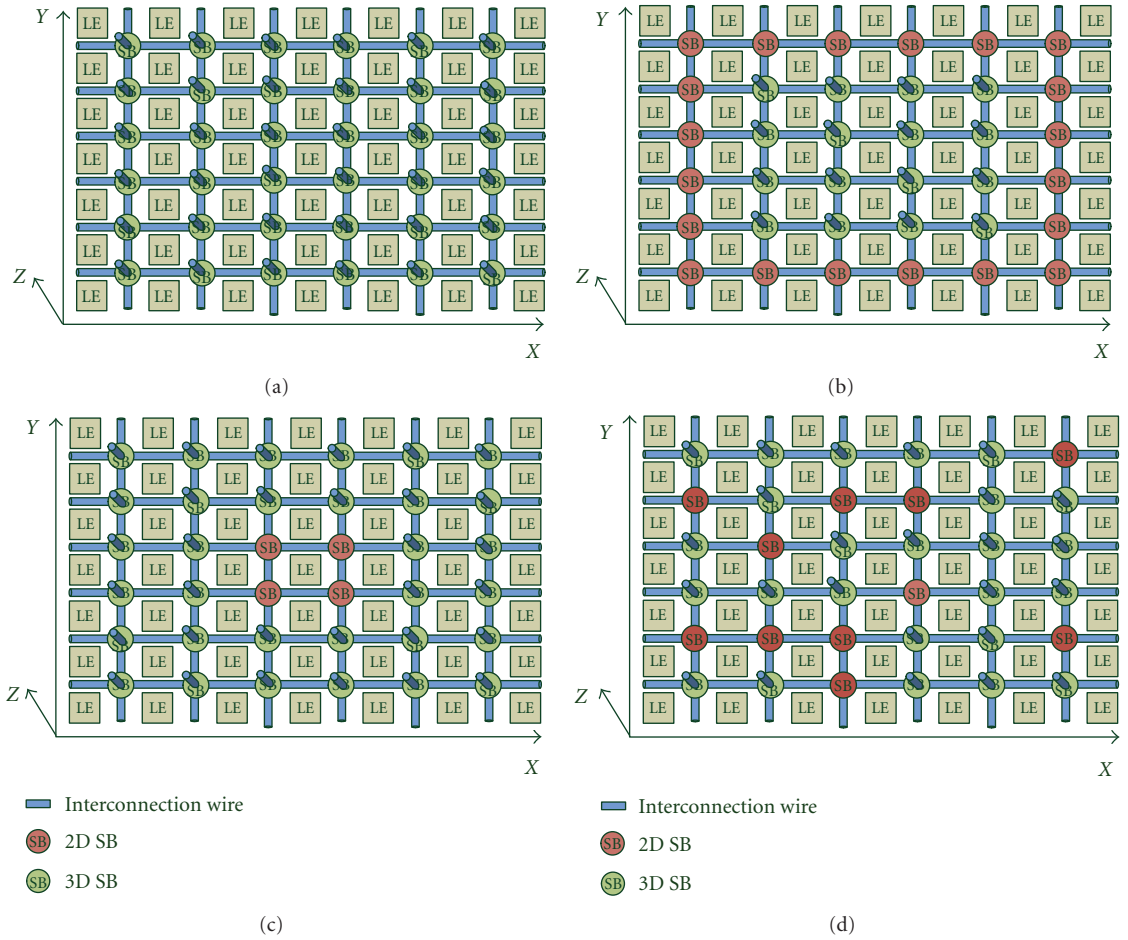


Figure 7: Alternative distribution scenarios for 3D SBs: (a) all the SBs are 3D, (b) the 3D SBs are assigned into the device center, (c) the 3D SBs are assigned into the device periphery, and (d) a full-custom assignment of 3D SBs.

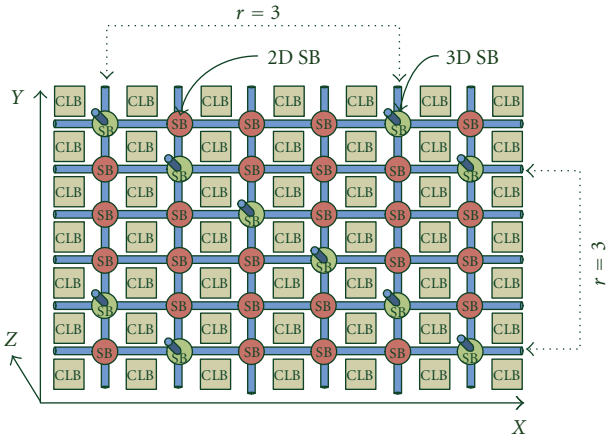


Figure 8: A layer from a 3D FPGA architecture with $r = 3$.

Such an interconnection architecture is based on a uniform distribution of 3D SBs across the device layers, which leads to significant improvement of routability problems.

In order to show the way that the 3D SBs are assigned, let us assume a layer of size $X \times Y$, where the percentage of 3D SBs are K% (over the total number of SBs exist in the layer). Then, the pattern of assigning these 3D SBs in each row and column of the layer is derived as follows: place a 3D SB to a spatial location (x, y) of a certain layer, then the neighboring 3D SBs across horizontal and vertical axis are assigned to the locations $(x + r, y, z)$ and $(x, y + r, z)$, respectively. In this expression, r indicates Manhattan distance between successive assigned 3D SBs (i.e., the number of 2D SBs between two neighboring 3D SBs). Due to the uniform distribution of vertical connections, the router of 3DPRO tool is assisted significantly in order to employ the vertical connections in a more efficient manner.

We have to stress that the combination of 2D and 3D SBs may result in a number of fabrication issues. More specifically, as the 3D SBs are larger compared to 2D SBs (due to the increased number of transistors), it is not obvious the way that these two types of SBs are combined in the same layout. For this purpose, our proposed methodology can support two solutions: (i) each 2D SB occupies area equal to the one required by a 3D SB, and (ii) to suppose that all the SBs are 3D but with a variation on the number of vertical interconnections (i.e., instead of changing the number of 2D/3D SBs, we change the number of TSVs per 3D SB). Both

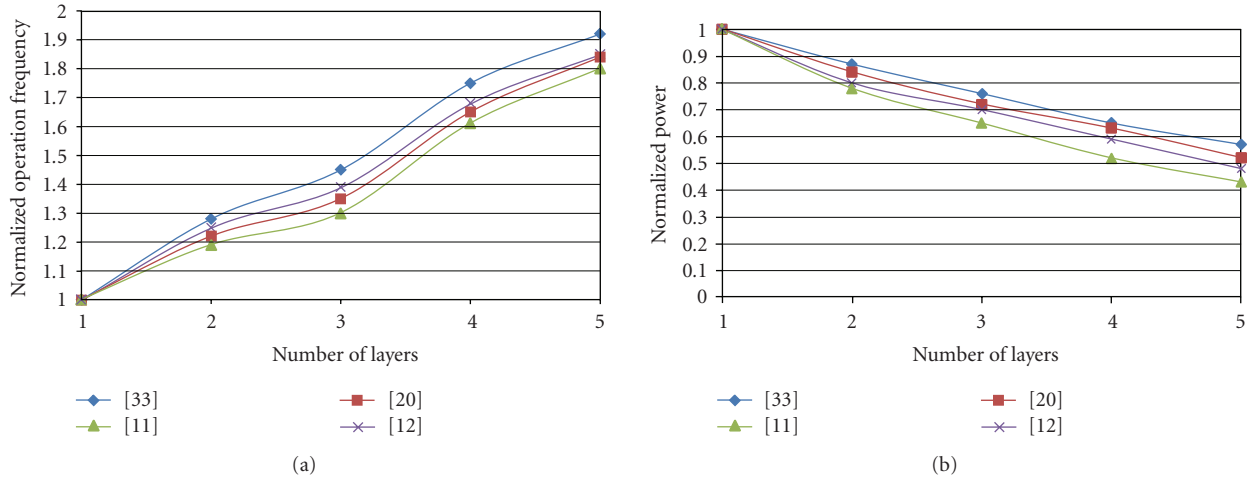


FIGURE 9: Average variation of application's delay and power consumption for a number of layers and TSVs with different electric characteristics.

of them is valid and might prevent the problem of nontrivial layouts of the regular structure shown in Figure 8.

Figures 9(a) and 9(b) plot the variation of maximum operation frequency and power consumption, respectively, over a number of different fabrication technologies for TSVs [11, 12, 20, 32], found in relevant references. We have to mention that among alternative process technologies, we employ the same number and distribution of TSVs. Even though more advanced TSVs might be found, the aim of Figure 9 is to illustrate that alternative fabrication processes for interlayer communication lead to significant improvement of critical design parameters (i.e., operation frequency and power consumption). More specifically, regarding the device with three functional layers, the performance improvement against the 2D FPGA balances between 20% and 40%, based on the selected electrical characteristics of the vertical connections. Similarly, such architecture reduces the power consumption compared to 2D FPGA from 22% up to 36%. These graphs concern the 3D architectures where all the SBs can form connections to the rest of the layers (i.e., 3D SBs).

Throughout this paper, the employed experimental setup for the targeted 3D-reconfigurable architectures can be summarized as follows:

(1) the 3D architectures consist of up to five functional layers;

(2) the hardware resources of each functional layer are identical. Based on this, both the amount of hardware resources (i.e., logic blocks, routing wires, TSVs, etc.) and their spatial location among layers are irrelative;

(3) the percentage of vertical interconnects (i.e., TSVs) per functional layer ranges from 10% up to 100%, with a step of 10%;

(4) each 3D SB realizes four vertical connections. In other words, each 3D SB placed on functional layer i has 4 TSVs to the layer $i-1$ and 4 other TSVs for the layer $i+1$. An exception to this occurs for the bottom and top layers of the 3D stack;

(5) the electrical parameters for each TSV correspond to fabrication technologies for 3D ICs found in relevant references.

The next figures show the average variation of some design parameters for 3D FPGAs with alternative interconnection scenarios regarding the TSVs distribution. For these graphs, the TSV's resistance is 350 mΩ, while its capacitance is 2.5 fF [12]. The horizontal axis corresponds to the percentage of fabricated TSVs on each layer, while the vertical one shows the normalized value of each design parameter (i.e., delay, power, Energy × Delay Product, etc.) in relation to a 2D FPGA. The percentage of TSVs for each layer corresponds to the number of 3D SBs placed on this layer, over the total number of SBs for this layer. However, (9) provides the mathematical expression for calculating this percentage. The architecture that corresponds to 100% fabricated TSVs per layer correspond to a 3D FPGA where every SB can form connections to the third dimension (similar to the TPR [2]):

$$\text{Percentage of 3D SBs} = \frac{\text{Number of 3D SBs per layer}}{\text{Total number of SBs (2D + 3D) per layer}} \times 100\%. \quad (9)$$

Figure 10 plots the average variation of Energy × Delay product (EDP) for the MCNC benchmarks [31], benchmarks for alternative 3D FPGAs. The normalization was performed over the EDP value of a conventional (i.e., 2D) FPGA. It can be seen that the increase of the number of layers results in more efficient realizations of the applications in 3D FPGAs. Also, we can claim that the proposed partitioning and P&R algorithms provide promising results for 3D architectures, where only a percentage of SBs forms 3D connections.

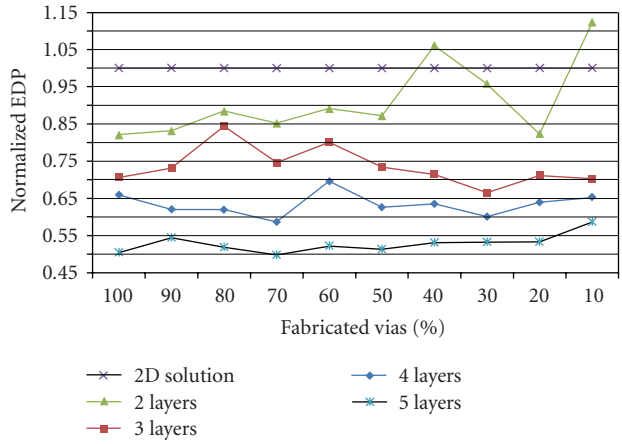


Figure 10: Average Energy×Delay Product (EDP) for different number of functional layers and percentage of fabricated TSVs.

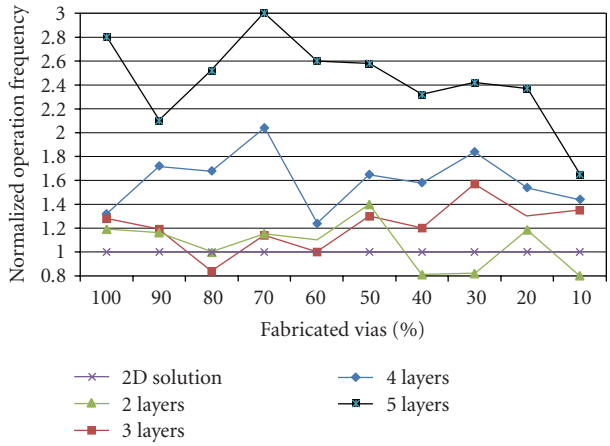


Figure 11: Average operation frequency over the MCNC benchmarks for different number of layers and percentages of fabricated TSVs.

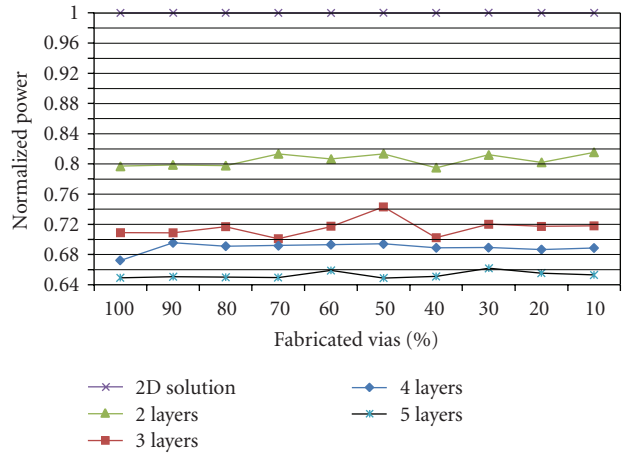


Figure 12: Average power consumption over the MCNC benchmarks for different number of functional layers and percentage of fabricated TSVs.

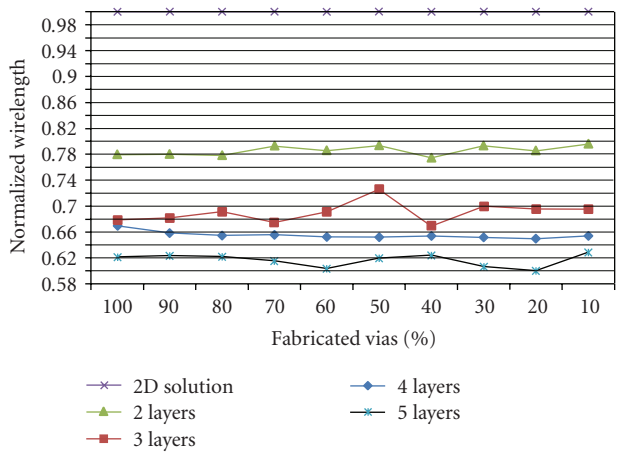


Figure 13: Average wirelength over the MCNC benchmarks for different number of functional layers and percentage of fabricated TSVs.

One of the main advantages of 3D integration is the increased operation frequency. Figure 11 plots the average variation of this parameter over the MCNC benchmarks for different number of layers and percentage of fabricated TSVs. The values in the vertical axis are normalized over the operation frequency that exhibits a 2D FPGA. Based on this graph, it is evident that the solution with five layers outperforms all the other implementations, achieving to increase the device operation frequency up to 30% (for percentage of fabricated vias 70%), as compared to 2D architectures. Moreover, by increasing the number of functional layers, the applications exhibit higher operation frequencies. However, it should be mentioned that for some architectures (i.e., those consisting of two layers), the operation frequency is low, as compared to the 2D solution, as the limited vertical interconnections stress the routing algorithm.

Due to the fact that reconfigurable devices exhibit high power dissipation, which is mainly occurred due to increased resistance/capacitance values exhibited by the interconnection network, we also study the total power requirements of the alternative 3D architectures. The results are summarized on Figure 12. Based on this graph, as the number of functional layers increases, the total power consumption reduced. Also, we can conclude that the 3D FPGA with five layers achieves to reduce the power consumption up to about 35%, as compared to 2D FPGA.

The gains in the performance and energy consumption depend on the intrinsic feature of 3D integration, regarding the minimization of wirelength. Among others, shorter wires lead to smaller resistance/capacitance, and hence to reduced delay, power/energy consumption, as well as silicon area footprint. Figure 13 shows the average wirelength requirements for different number of functional layers and percentages of fabricated TSVs over the MCNC benchmarks. As the number of functional layers increases, the total wirelength is reduced due to the increased number of neighbors. Also, the wirelength reduction from 2D FPGAs to a 3D stack with two function layers is about

Table 3: Comparison results between MCNC benchmarks: implementation in 2D and 3D FPGA architecture with three functional layers as well as $K = 30\%$ and $K = 100\%$ TSVs.

Benchmark	Wirelength ($\times 10^3$)			Delay ($\times 10^{-9}$ sec)			Power ($\times 10^{-3}$ Watt)		
	2D	3D $K = 30\%$	3D $K = 100\%$	2D	3D $K = 30\%$	3D $K = 100\%$	2D	3D $K = 30\%$	3D $K = 100\%$
alu4	37.81	35.65	37.09	73.7	45.9	47.2	115.93	67.22	73.52
apex2	58.99	56.06	53.16	105	53.4	50.5	106.99	52.91	54.69
apex4	37.76	38.16	37.92	79.2	43.7	41.1	068.38	38.91	34.77
bigkey	32.71	48.57	43.68	38.1	23.0	22.2	317.79	180.60	172.70
clma	430.44	294.60	280.50	170	114	103	456.00	323.02	283.89
des	52.70	45.44	43.82	63.4	43.3	39.2	231.04	142.05	158.61
diffeq	34.88	36.31	31.08	58.8	60.3	60.4	104.14	97.05	116.54
dsip	29.90	35.46	33.69	31.9	22.0	20.2	349.29	235.90	194.56
elliptic	102.24	92.15	94.91	89.1	92.6	83.7	272.58	280.38	266.23
ex1010	36.90	33.78	31.47	54.0	47.7	46.7	103.95	83.44	90.77
ex5p	129.65	167.00	146.42	163	73.5	73.8	117.65	46.71	46.37
frisc	174.05	99.83	91.26	100	110	105	152.35	160.13	163.65
misex3	39.31	38.26	37.88	86.9	40.3	46.4	86.93	41.12	41.80
pdc	238.78	173.32	160.37	153	79.7	78.3	179.82	86.11	82.73
s298	55.85	44.65	42.30	187	92.5	87.1	78.68	36.53	41.72
s38417	172.01	169.52	155.97	90.2	64.1	74.8	284.25	185.84	25.92
s38584	129.14	152.62	136.39	97.3	60.8	55.6	264.54	183.45	129.86
Seq	52.80	54.49	48.74	66.6	47.6	44.8	132.37	105.93	89.05
Spla	148.19	113.47	125.03	132	64.8	66.4	125.79	64.58	61.77
Tseng	25.89	23.85	21.07	63.9	54.8	55.5	93.76	71.52	71.60
Average	101.00	87.659	82.64	95.2	61.7	60.1	182.00	124.00	122.00
Ratio	1.00	0.87	0.82	1.00	0.65	0.63	1.00	0.68	0.67

22%, while if we increase the number of layers to five, then the additional reduction to total wirelength is only 18%.

Several points can be made from the previous graphs/plots. Among others, as we increase the number of layers, the applications are realized more efficiently in 3D FPGAs compared to 2D-reconfigurable architectures. Secondly, we can claim that the proposed algorithms provide promising results for 3D architectures, where only a percentage of SBs forms connections to the third dimension. More specifically, we can conclude that as we vary the number of fabricated TSVs on each layer, significant reduction on design parameters may be achieved, leading to more efficient 3D architectures.

It is worth mentioning that in contrast to Figure 9, where the increase of number of layers leads to monotonous gains in performance and power consumption, this is not valid for Figures 10–13. In these graphs, we also study the impact of different amount and distribution of 3D SBs over the device layers. More specifically, as we modify the percentage of 3D SBs over the total number of SBs placed onto a layer, we alter the connectivity resource graph (i.e., the graph that describes the routing resources of the target device). Due to this, the routing algorithm has to pay effort in order to find new paths to connect the logic blocks. As these paths do not exhibit same lengths, they have significant variations on the RC parameters. The employed models both for estimating delay [33] and power [24] are related to three main parameters of wires, namely, their length, resistance, and capacitance. The nonmonotonous form of these curves is due to these variations of the resistance/capacitance and wirelength for the routing paths.

The nonmonotonous behavior of these curves (for given the number of layers) can be explained as follows: each of these curves shows the impact of alternative 3D architectures, which occupy different percentage of TSVs. More specifically, as we modify the percentage of 3D SBs over the total number of SBs placed onto a layer, we alter the connectivity resource graph (i.e., the graph that describes the routing resources of the target device). Due to this, the routing algorithm has to pay effort in order to find new paths to connect the logic blocks. As these paths do not exhibit same lengths, they have significant variations on the RC parameters. The employed models both for estimating delay [33] and power [24] are related to three main parameters of wires, namely, their length, resistance, and capacitance. Consequently, the variations of these parameters results in the nonmonotonous behavior of these architecture solutions. Also, we have to mention that for each number of layers, these graphs do not show curves, but distinct architecture solutions regarding devices with specific percentages of TSVs. For demonstration purposes, we have just connected dots with lines in order to show the variation on 3D efficiency as we alter the amount of interlayer connections.

Table 4: Comparison results between 20 biggest MCNC benchmarks: via utilization in 3D FPGA architecture (with 30% and 100% via links, 3 layers, and maxima operation frequency).

Benchmark	30% 3D SBs			100% 3D SBs		
	Total vias (fabricated)	Actually utilized vias	(%)	Total vias (fabricated)	Actually utilized vias	(%)
alu4	2799	1148	41%	10109	3639	36%
apex2	3456	1140	33%	9600	4512	47%
apex4	2705	1190	44%	6242	2185	35%
Bigkey	3379	1385	41%	7798	3119	40%
Clma	17781	7290	41%	46570	19559	42%
Des	2540	813	32%	11642	5123	44%
Diffeq	2289	847	37%	7630	2365	31%
Dsip	3266	1143	35%	10886	3484	32%
Elliptic	6823	2661	39%	19246	8468	44%
ex1010	6919	2491	36%	23064	10609	46%
ex5p	2705	1353	50%	9710	4370	45%
frisk	5841	1752	30%	16224	6003	37%
misex3	4032	1976	49%	9600	3840	40%
Pdc	5774	2021	35%	22745	10918	48%
s298	2580	903	35%	5530	2488	45%
s38417	7776	3577	46%	25920	12182	47%
s38584	8995	3418	38%	29983	13492	45%
Seq	3639	1674	46%	7798	3197	41%
Spla	5391	1941	36%	16589	6636	40%
Tseng	1903	564	30%	6344	3900	61%
Average	5030	1964	39%	15161	6504	43%
Ratio	1.00	0.39		1.00	0.43	

In order to evaluate the 3D FPGAs with reduced number of vertical connections, we provide some experimental results regarding the P&R on devices consisting of 3 functional layers with different percentage of fabricated TSVs (shown in Table 3). More specifically, we evaluate a 3D FPGA with $K = 30\%$ of the TSVs fabricated against a device with identical functional layers in each layer but with TSVs in every SBs (i.e., $K = 100\%$). For the sake of completeness, we provide also the metrics regarding the 2D FPGA. The percentage selection of 30% is retrieved from Algorithm 2 due to the minimum of EDP curve for a 3D device with three functional layers. We evaluate the alternative architectures in terms of wirelength, delay, and power consumption. Comparing with 2D devices, the above-mentioned results prove that the 3D architectures provide significant reduction in wirelength, delay, and power consumption.

Considering the percentage of fabricated vias equal to 30%, the average reduction in the wirelength, the delay, and the power consumption is 13%, 35%, and 33%, respectively. Similarly, the corresponding values for 100% vias are 18%, 27%, and 33%, respectively. Indeed, the wirelength reduction (i.e., resistance and capacitance reduction), due to 3D integration, results in remarkable improvements in delay and energy consumption. However, these savings seem to be independent of the number of fabricated TSVs, as the 3D device with $K = 100\%$ achieves almost similar gains compared to the one with $K = 30\%$. The proposed methodology for eliminating the number of vertical connections has a penalty in the number of routing tracks of each layer. The average increase of them is about 8%. However, the solution with less fabricated TSVs is more reliable (due to technology parameters) and cost-efficient. Also, with the current process technologies, it is almost impossible to fabricate such a high amount of vertical interconnections. Due to this, the proposed approach derives the gains of the 3D integration with a more feasible technology approach. The extra penalty in layer's area (due to the increased amount of routing wires) cannot outperform the benefits of 3D FPGAs, as compared to conventional 2D solutions. To the best to our knowledge, it is the first time in the literature where the efficiency of a 3D FPGA architecture remains unchanged with less hardware resources (i.e., fewer vias).

More details about the TSVs utilization on the 20 biggest MCNC benchmarks can be found on Table 4. As we can concluded, the percentage of utilized vias for three-layer FPGA architectures does not depend on the percentage of fabricated vertical links (vias) between layers. More specifically, the average utilization ratio of vias for FPGAs composed by 30% and 100% 3D SBs (vias) are 39% and 43%, respectively. Consequently, we proved that the design of efficient 3D FPGA architectures with smaller number of vias than 100% is feasible with reduced fabrication costs. In contrary, all the existing designs support "fully-populated" TSVs 3D FPGA designs only.

The last point is very important because we manage to achieve the same improvements employing less hardware

resources (i.e., TSVs). More specifically, from 3D fabrication/manufacturing point of view, the smaller number of vertical connections means: (i) smaller fabrication costs and (ii) larger useful silicon area in each layer (a TSV contact occupies much more silicon area than a simple metal contact). The increased connectivity in the vertical axis means more silicon and eventually greater cost. Even though there are also two other known P&R tools, however, we could not provide sufficient comparison results against them. More specifically, the approach in [2] does not provide any estimation regarding the power consumption, while the one in [3] is not publicly available. Additionally, both of them assume "fully populated" TSVs 3D FPGA devices only (scenario $K = 100\%$).

6. Conclusions

A systematic software-supported methodology for exploring and evaluating alternative interconnection schemes for 3D FPGAs is presented. The methodology is supported by three new CAD tools (part of the 3D MEANDER Design Framework). The evaluation results prove that it is possible to design 3D FPGAs with limited number of vertical connections without any penalty in performance or power consumption. More specifically, for the 20 biggest MCNC benchmarks, the average gains in operation frequency, total wirelength, and energy consumption are 35%, 13%, and 32%, respectively, compared to existing 2D FPGAs with identical logic resources.

Acknowledgments

This paper is part of the 03ED593 research project, implemented within the framework of the Reinforcement Program of Human Research Manpower (PENED) and cofinanced by National and Community Funds (75% from EU.-European Social Fund and 25% from the Greek Ministry of Development: General Secretariat of Research and Technology).

References

[1] E. Beyne, "3D interconnection and packaging: impending reality or still a dream?" in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '04)*, vol. 1, pp. 138–139, San Francisco, Calif, USA, February 2004.

[2] C. Ababei, Y. Feng, B. Goplen, et al., "Placement and routing in 3D integrated circuits," *IEEE Design & Test of Computers*, vol. 22, no. 6, pp. 520–531, 2005.

[3] S. Das, A. Fan, K.-N. Chen, C. S. Tan, N. Checka, and R. Reif, "Technology, performance, and computer-aided design of three-dimensional integrated circuits," in *Proceedings of International Symposium on Physical Design*, pp. 108–115, Phoenix, Ariz, USA, April 2004.

[4] A. Rahman, S. Das, A. P. Chandrakasan, and R. Reif, "Wiring requirement and three-dimensional integration technology for field programmable gate arrays," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 1, pp. 44–54, 2003.

[5] K. Siozios, K. Sotiriadis, V. F. Pavlidis, and D. Soudris, "A software-supported methodology for designing high-performance 3D FPGA architectures," in *Proceedings of IFIP International Conference on Very Large Scale Integration (VLSI-SoC '07)*, pp. 54–59, Atlanta, Ga, USA, October 2007.

[6] C. H. Yu, "The 3rd dimension—more life for Moore's Law," in *Proceedings of International Microsystems, Packaging, Assembly Conference Taiwan (IMPACT '06)*, pp. 1–6, Taipei, Taiwan, October 2006.

[7] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex™-II FPGA family," in *Proceedings of the 10th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '02)*, pp. 157–164, Monterey, Calif, USA, February 2002.

[8] D. Wang, Y. Xie, Y. Hu, H. Li, and X. Li, "Hierarchical fault tolerance memory architecture with 3-dimension interconnect," in *Proceedings of the 10th IEEE Region Annual International Conference (TENCON '07)*, pp. 1–4, Taipei, Taiwan, October-November 2007.

[9] R. Reif, A. Fan, K.-N. Chen, and S. Das, "Fabrication technologies for three-dimensional integrated circuits," in *Proceedings of the 3rd International Symposium on Quality Electronic Design (ISQED '02)*, pp. 33–37, San Jose, Calif, USA, March 2002.

[10] V. F. Pavlidis and E. G. Friedman, "Interconnect delay minimization through interlayer via placement in 3-D ICs," in *Proceedings of the 15th ACM Great Lakes Symposium on VLSI*, pp. 20–25, Chicago, Ill, USA, April 2005.

[11] A. W. Topol, D. C. La Tulipe Jr., L. Shi, et al., "Three-dimensional integrated circuits," *IBM Journal of Research and Development*, vol. 50, no. 4-5, pp. 491–506, 2006.

[12] S. Gupta, M. Hilbert, S. Hong, and R. Patti, "Techniques for producing 3D ICs with high-density interconnect," in *Proceedings of the 21st International VLSI Multilevel Interconnection Conference (VMIC '04)*, Waikoloa Beach, Hawaii, USA, September-October 2004.

[13] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[14] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3D-FPGA," in *Proceedings of the 14th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '06)*, pp. 113–122, Monterey, Calif, USA, February 2006.

[15] http://vlsi.ee.duth.gr/amdrel.

[16] K. Siozios, G. Koutroumpezis, K. Tatas, et al., "A novel FPGA architecture and an integrated framework of CAD tools for implementing applications," *IEICE Transactions on Information and Systems*, vol. E88-D, no. 7, pp. 1369–1380, 2005.

[17] K. Siozios, K. Tatas, G. Koutroumpezis, D. Soudris, and A. Thanailakis, "An integrated framework for architecture level exploration of reconfigurable platform," in *Proceedings of the 15th International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 658–661, Tampere, Finland, August 2005.

[18] A. Rahman, J. Trezza, B. New, and S. Trimberger, "Die stacking technology for terabit chip-to-chip communications," in *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 587–590, San Jose, Calif, USA, September 2006.

[19] F. M. Finkbeiner, C. Adams, E. Apodaca, et al., "Development of ultra-low impedance through-wafer Micro-vias," *Nuclear Instruments and Methods in Physics Research Section A*, vol. 520, no. 1–3, pp. 463–465, 2004.

[20] S. M. Alam, R. E. Jones, S. Rauf, and R. Chatterjee, "Inter-strata connection characteristics and signal transmission in three-dimensional (3D) integration technology," in *Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED '07)*, pp. 580–585, San Jose, Calif, USA, March 2007.

[21] D. M. Jang, C. Ryu, K. Y. Lee, et al., "Development and evaluation of 3-D SiP with vertically interconnected Through Silicon Vias (TSV)," in *Proceedings of the 57th Electronic Components and Technology Conference (ECTC '07)*, pp. 847–852, Reno, Nev, USA, May-June 2007.

[22] I. Das and J. E. Dennis, "Normal-boundary intersection: a new method for generating the Pareto surface in nonlinear multicriteria optimization problems," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 631–657, 1998.

[23] N. Selvakkumaran and G. Karypis, "Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 504–517, 2006.

[24] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 2, pp. 279–302, 2005.

[25] K. Siozios, K. Tatas, G. Koutroumpezis, D. Soudris, and A. Thanailakis, "An integrated framework for architecture level exploration of reconfigurable platform," in *Proceedings of the 15th International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 658–661, Tampere, Finland, August 2005.

[26] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: a survey," *Integration, the VLSI Journal*, vol. 19, no. 1-2, pp. 1–81, 1995.

[27] W. Swartz and C. Sechen, "New algorithms for the placement and routing of macro cells," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '90)*, pp. 336–339, Santa Clara, Calif, USA, November 1990.

[28] J. Lam and J.-M. Delosme, "Performance of a new annealing schedule," in *Proceedings of the 25th ACM/IEEE Design Automation Conference (DAC '88)*, pp. 306–311, Atlantic City, NJ, USA, June 1988.

[29] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *Proceedings of International Conference on Computer-Aided Design (ICCAD '86)*, pp. 381–384, Santa Clara, Calif, USA, October 1986.

[30] L. McMurchie and C. Ebeling, "PathFinder: a negotiation-based performance-driven router for FPGAs," in *Proceedings of the 3rd International ACM Symposium on Field Programmable Gate Arrays (FPGA '95)*, pp. 111–117, Monterey, Calif, USA, February 1995.

[31] "Standard Cell Benchmark Circuits from the Microelectronics Center of North Carolina (MCNC)," http://vlsicad.cs.binghamton.edu/gz/PDWorkshop91.tgz.

[32] L. L. W. Leung and K. J. Chen, "Microwave characterization and modeling of high aspect ratio through-wafer interconnect vias in silicon substrates," *IEEE Transactions on Microwave Theory and Techniques*, vol. 53, no. 8, pp. 2472–2480, 2005.

[33] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '96)*, pp. 44–49, San Jose, Calif, USA, November 1996.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 738174, 17 pages
doi:10.1155/2008/738174

Research Article

Multiobjective Optimization for Reconfigurable Implementation of Medical Image Registration

Omkar Dandekar,[1,2] William Plishker,[1,2] Shuvra S. Bhattacharyya,[1] and Raj Shekhar[1,2]

[1] *Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742, USA*
[2] *Department of Diagnostic Radiology and Nuclear Medicine, University of Maryland School of Medicine, Baltimore, MD 21201, USA*

Correspondence should be addressed to Raj Shekhar, rshekhar@umm.edu

Received 6 March 2008; Revised 11 September 2008; Accepted 27 November 2008

Recommended by Juergen Becker

In real-time signal processing, a single application often has multiple computationally intensive kernels that can benefit from acceleration using custom or reconfigurable hardware platforms, such as field-programmable gate arrays (FPGAs). For adaptive utilization of resources at run time, FPGAs with capabilities for dynamic reconfiguration are emerging. In this context, it is useful for designers to derive sets of efficient configurations that trade off application performance with fabric resources. Such sets can be maintained at run time so that the best available design tradeoff is used. Finding a single, optimized configuration is difficult, and generating a family of optimized configurations suitable for different run-time scenarios is even more challenging. We present a novel multiobjective wordlength optimization strategy developed through FPGA-based implementation of a representative computationally intensive image processing application: medical image registration. Tradeoffs between FPGA resources and implementation accuracy are explored, and Pareto-optimized wordlength configurations are systematically identified. We also compare search methods for finding Pareto-optimized design configurations and demonstrate the applicability of search based on evolutionary techniques for identifying superior multiobjective tradeoff curves. We demonstrate feasibility of this approach in the context of FPGA-based medical image registration; however, it may be adapted to a wide range of signal processing applications.

1. Introduction

In the field of real-time signal processing systems, acceleration of computationally intensive algorithmic components is often achieved by mapping them to custom or reconfigurable hardware platforms, such as field-programmable gate arrays (FPGAs). Often, multiple kernels in a single application can benefit from this approach to acceleration, requiring them to share a single fabric. This is particularly necessary in applications where multiple kernels share data and feed results to each other. For example, in medical imaging it has been shown that both image preprocessing [1–3] and image registration [4–6] can achieve high levels of speedup through hardware acceleration. To maximize the performance of an application and to optimize the fabric resource utilization, the kernels must be designed to meet their application requirements while balancing their resource consumption on the fabric. Application requirements often change at run time and strategies based on static design must try to identify a reasonable "average case" design configuration that accommodates all possible scenarios. Because this approach can be highly suboptimal and can result in significant under- or overutilization of the fabric in many scenarios, modern FPGAs are emerging with run-time reconfiguration capabilities. Self-monitoring FPGA implementations are able to adapt to variable application requirements and reconfigure their processing structures to better-suited design configurations [7]. This not only improves application performance but also results in more effective utilization of fabric resources. To exploit this technology, it is highly desirable that the designers provide quality design configurations that trade off application performance with fabric resources. Consequently, the primary focus of this work is to develop a framework that enables the designers to identify such optimized design configurations.

A common system parameter for trading off resource and performance is datapath wordlength. Typically, algorithms are first developed in software using floating-point representation and later migrated to hardware using finite precision (e.g., fixed-point representation) for achieving improved computational speed and reduced hardware cost. These implementations are often parameterized, so that a wide range of finite precision representations can be supported [8] by choosing an appropriate wordlength for each internal variable. As a consequence, the accuracy and hardware resource requirements of such a system are functions of the wordlengths used to represent the internal variables. Determining an optimal wordlength configuration that minimizes the hardware implementation cost while satisfying a design criterion such as maximum output error has been shown to be nondeterministic polynomial-time (NP)-hard [9] and can take up to 50% of the design time for complex systems [10]. In addition, a single optimal solution may not exist, especially in the presence of multiple conflicting objectives. Moreover, a new configuration generally must be derived when the design constraints are altered.

An optimum wordlength configuration can be identified by analytically solving the quantization error equation as described previously by several authors [11–15]. This analytical representation, however, can be difficult to obtain for complex systems. Techniques based on local search or gradient-based search [16] have also been employed, but these methods are limited to finding a single feasible solution as opposed to an optimized tradeoff curve. An exhaustive search of the entire design space is guaranteed to find Pareto-optimal configurations. Execution time for such exhaustive search, however, increases exponentially with the number of design parameters, making it unfeasible for most practical systems. Methods that transform this problem into a linear programming problem have also been reported [11], but these techniques are limited to cases in which the objectives can be modeled as linear functions of the design parameters. Other approaches based on linear aggregation of objectives may not find proper Pareto-optimal solutions when the search space is nonconvex [17]. Techniques based on evolutionary methods have been shown to be effective in searching large search spaces in an efficient manner [18, 19]. Furthermore, these techniques are inherently capable of performing multipoint searches. As a result, techniques based on evolutionary algorithms (EAs) have been employed in the context of multiobjective optimization (SPEA2 [20], NSGA-II [21]). However, their application to solving wordlength optimization problems has been limited.

We formulate this problem of finding optimal wordlength configurations as a multiobjective optimization, where different objectives—for example, accuracy and area—generally conflict with one another. Although this approach increases the complexity of the search, it can find a set of Pareto-optimized configurations representing strategically chosen tradeoffs among the various objectives. This allows a designer to choose an efficient configuration that satisfies given design constraints and provides ease and flexibility in modifying the design configuration as the constraints change. In this work, we present this novel multiobjective optimization strategy and demonstrate its feasibility in the context of FPGA-based implementation of medical image registration. The tradeoff between FPGA resources (area and memory) and implementation accuracy is systematically explored, and Pareto-optimized solutions are identified. This analysis is performed by treating the wordlengths of the internal variables as design variables. We also compare several search methods for finding Pareto-optimized solutions and demonstrate, in the context of the chosen problem, the applicability of search based on evolutionary techniques for efficiently identifying superior multiobjective tradeoff curves. In comparison with the earlier reported techniques, our work captures more comprehensively the complexity of the underlying multiobjective optimization problem and demonstrates the applicability of our framework in finding superior Pareto-optimized solutions in an efficient manner, even in the presence of a nonlinear objective function.

This paper is organized as follows. Section 2 provides background on image registration and outlines an architecture for its FPGA-based implementation. We also highlight some strategies for parameterized design and synthesis of this architecture. Formulations for multiobjective optimization and various search methods to find Pareto-optimized solutions are described in Section 3. Section 4 describes experimental results, compares various search methods, and presents postsynthesis validation of the presented strategy. In Section 5, discussion on wordlength search and multiobjective optimization is presented. Section 6 concludes the paper.

2. Image Registration

Medical image registration is the process of aligning two images that represent the same anatomy at different times, from different viewing angles, or using different imaging modalities. Image registration is an active area of research, and over the last several decades numerous publications have outlined various methodologies to perform image registration and its applications. Maintz and Viergever [22] and Hill et al. [23] have presented a comprehensive summary of the range of the image registration domain. Several types of image registration are in routine use (see [22–25]); however, registration based on voxel intensities remains the most versatile, powerful, and inherently automatic way of achieving the alignment between two images. This approach, in general, attempts to find the transformation ($\hat{T}$) that optimally aligns a reference image (RI) with coordinates x, y, and z and a floating image (FI) under an image similarity measure ($\mathbb{F}$):

$$\hat{T} = \arg\max_{T} \mathbb{F}\left(\mathrm{RI}(x, y, z), \mathrm{FI}\left(T(x, y, z)\right)\right). \tag{1}$$

Many image similarity measures, such as the sum of squared differences and cross-correlation, have been used, but over the last decade mutual information (MI) has emerged as the preferred similarity measure. MI is an information theoretic measure and is calculated as:

$$\mathrm{MI}(\mathrm{RI}, \mathrm{FI}) = h(\mathrm{RI}) + h(\mathrm{FI}) - h(\mathrm{RI}, \mathrm{FI}). \tag{2}$$

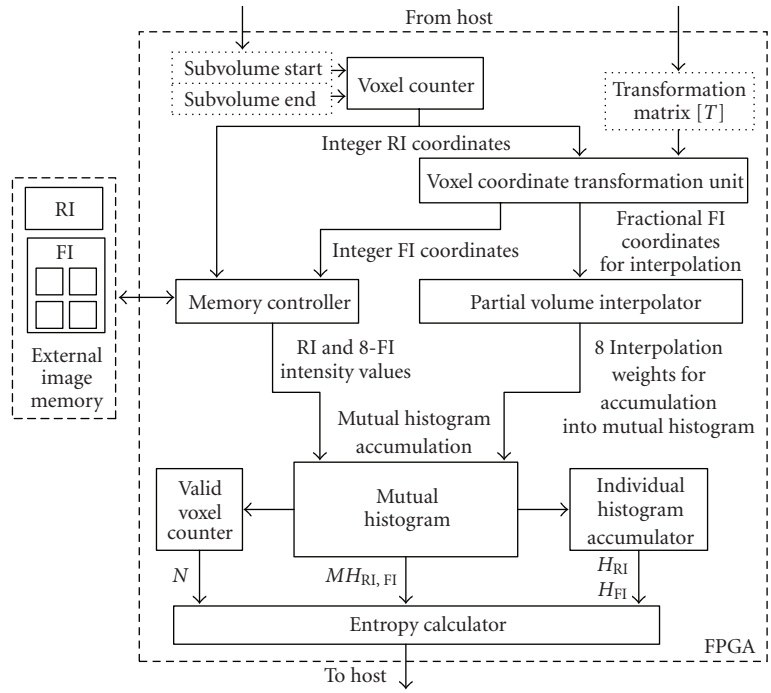


Figure 1: Top-level block diagram of FPGA-based architecture for MI calculation.

In this equation, $h(\mathrm{RI})$ and $h(\mathrm{FI})$ are the individual entropies and $h(\mathrm{RI}, \mathrm{FI})$ is the mutual entropy of the images to be registered. These entropies are further calculated as:

$$\begin{aligned} h(\mathrm{RI}) &= -\sum p_{\mathrm{RI}}(x) \cdot \ln\left(p_{\mathrm{RI}}(x)\right), \\ h(\mathrm{FI}) &= -\sum p_{\mathrm{FI}}(x) \cdot \ln\left(p_{\mathrm{FI}}(x)\right), \\ h(\mathrm{RI}, \mathrm{FI}) &= -\sum \sum p_{\mathrm{RI,FI}}(x) \cdot \ln\left(p_{\mathrm{RI,FI}}(x)\right). \end{aligned} \tag{3}$$

Here, the notations p_{RI}, p_{FI}, and $p_{\mathrm{RI,FI}}$ represent the individual probability distribution function (PDF) of RI, individual PDF of FI, and the mutual PDF of RI and FI, respectively. These distributions are estimated from the individual and mutual histograms of the images to be registered. Additional details about computation of MI, its properties, and its application to image registration can be found in an article by Pluim et al. [25].

MI-based image registration has been shown to be robust and effective in multimodality image registration [24]. However, this form of registration typically requires thousands of iterations (MI evaluations), depending on image complexity and the degree of initial misalignment between images. Castro-Pareja et al. [4] have shown that calculation of MI for different candidate transformations is a factor limiting the performance of MI-based image registration. We have, therefore, developed an FPGA-based architecture for accelerated calculation of MI [6] that is capable of computing MI 40 times faster than software implementation. The transformation model (T in (1)) employed by this architecture is a locally rigid-body model consisting of three-dimensional (3D) translations and rotations. Consequently, the analysis presented in this article pertains to locally rigid transformations. However, it must be noted that hierarchical rigid-body transformations can be used to represent deformable (nonrigid) transformation models as demonstrated in the volume subdivision-based approach reported by Walimbe and Shekhar [26].

2.1. FPGA-Based Implementation of Mutual Information Calculation

During the execution of image registration using this architecture, the optimization process is executed from a host workstation. The host provides a candidate transformation, while the FPGA-based implementation applies it to the images and performs the corresponding MI computation. The computed MI value is then further used by the host to update the candidate transformation and eventually find the optimal alignment between the RI and FI. Figure 1 shows the top-level block diagram of the aforementioned architecture. The important modules in this design are described in the following sections.

2.1.1. Voxel Counter

Calculation of MI requires processing (fetching the voxel from the image memory, performing coordinate transformation, and updating the mutual histogram (MH)) each voxel in the RI. In addition, because the implemented algorithm processes the images on a subvolume basis, RI voxels within a 3D neighborhood corresponding to an individual subvolume must be processed sequentially. The host programs the FPGA-based MI calculator with subvolume start and end

addresses, and the voxel counter computes the address corresponding to each voxel within that subvolume in *z-y-x* order.

2.1.2. Coordinate Transformation

The initial step in MI calculation involves applying a candidate transformation (T) to each voxel coordinate ($\vec{v}_r$) in the RI to find the corresponding voxel coordinates in the FI ($\vec{v}_f$). This is mathematically expressed as:

$$\vec{v}_f = T \cdot \vec{v}_r. \tag{4}$$

The deformation model employed is a six-parameter rigid transformation model and is represented using a 4×4 matrix. The host calculates this matrix based on the current candidate transformation provided by the optimization routine and sends it to the MI calculator. A fixed-point representation is used to store the individual elements of this matrix. The coordinate transformation is accomplished by a simple matrix multiplication.

2.1.3. Partial Volume Interpolation

The coordinates mapped in the FI space ($\vec{v}_f$) do not normally coincide with a grid point (integer location), thus requiring interpolation. Nearest neighbor and trilinear interpolation schemes have been used most often for this purpose; however, partial volume (PV) interpolation, introduced by Maes et al. [24], has been shown to provide smooth changes in the histogram values with small changes in transformation. The reported architecture consequently implements PV interpolation as the choice of interpolation scheme. $\vec{v}_f$, in general, will have both fractional and integer components and will land within an FI neighborhood of size $2 \times 2 \times 2$. The interpolation weights required for the PV interpolation are calculated using the fractional components of $\vec{v}_f$. Fixed-point arithmetic is used to compute these interpolation weights. The corresponding floating voxel intensities are fetched by the image controller in parallel using the integer components of $\vec{v}_f$. The image controller also fetches the voxel intensity corresponding to $\vec{v}_r$. The MH then must be updated for each pair of reference and floating voxel intensities (eight in all) using the corresponding weights computed by the PV interpolator.

2.1.4. Image Memory Access

Images from different modalities (computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), etc.) have different native resolution, typical image dimensions, and dynamic range. Despite these variations, dimensions of most medical images are smaller than $512 \times 512 \times 512$ and are supported by our architecture. The dynamic range of these images is indicated by the number of bits used to represent the intensity (gray value) at every voxel. For MI-based registration, however, these images are typically converted to 7- or 8-bit representation as a part of image preprocessing. This is done to prevent dispersion of the MH and leads to improved quality of image registration. After this preprocessing step, all the gray values in the images are used for image registration.

The typical size of 3D medical images prevents the use of high-speed memory internal to the FPGA for their storage. Between the two images, the RI has more relaxed access requirements because it is accessed in a sequential manner (in *z-y-x* order). This kind of access benefits from burst accesses and memory caching techniques, allowing the use of modern dynamic random access memories (DRAMs) for image storage. For the architecture presented, both the RI and FI are stored in separate logical partitions of the same DRAM module. Because the access to the RI is sequential and predictable, the architecture uses internal memory to cache a block of RI voxels. Thus, during the processing of that block of RI voxels, the image controller has parallel access to both RI and FI voxels. The RI voxels are fetched from the internal FPGA memory, whereas the FI voxels are fetched directly from the external memory.

The FI, however, must be accessed randomly (depending on the current transformation T), and eight FI voxels (a $2 \times 2 \times 2$ neighborhood) must be fetched for every RI image voxel to be processed. To meet this memory access requirement, the reported architecture employs a memory addressing scheme similar to the cubic addressing technique reported in the context of volume rendering [27] and image registration [4]. A salient feature of this technique is that it allows simultaneous access to the entire $2 \times 2 \times 2$ voxel neighborhood. The reported architecture implements this technique by storing four copies of the FI and taking advantage of the burst mode accesses native to modern DRAMs. The image voxels are arranged sequentially such that performing a size-two burst fetches two adjacent 2×2 neighborhood planes, thus making the entire neighborhood available simultaneously. The image intensities of this neighborhood are then further used for updating the MH.

2.1.5. Updating the Mutual Histogram

For a given RI voxel (RV) and associated 3D neighborhood in the FI ($FV_0 : FV_7$), there are eight intensity pairs ($RV, FV_0 : FV_7$) and corresponding interpolation weights. Because the MH must be updated (read-modify-write) at these eight locations, this amounts to 16 accesses to MH memory for each RI voxel. This high memory access requirement is handled by using the high-speed, dual-ported memories internal to the FPGA to store the MH. The operation of updating the MH is pipelined and, hence, read-after-write (RAW) hazards can arise if consecutive transactions attempt to update identical locations within the MH. The reported design addresses this issue by introducing preaccumulate buffers, which aggregate the weights from all conflicting transactions. Thus, all the transactions leading to an RAW hazard are converted into a single update to the MH, thereby eliminating any RAW hazards.

While the MH is being computed, the individual histogram accumulator unit computes the histograms for the RI and FI. These individual histograms are also stored

using internal, dual-ported memories. The valid voxel counter module keeps track of the number of valid voxels accumulated in the MH and calculates its reciprocal value. The reciprocal value of the number of valid voxels in the histogram is calculated by using successive subtraction operations. This operation takes N clock cycles (where N is the fractional wordlength of the reciprocal value) and must be performed only once per every MI calculation. The resulting value is then used by the entropy calculation unit for calculating the individual and joint probabilities and subsequently entropies as described in (3).

2.1.6. Entropy Calculation

The final step in MI calculation is to compute joint and individual entropies using the joint and individual probabilities, respectively. To calculate entropy, it is necessary to evaluate the function $f(p) = p \cdot \ln(p)$ for all the probabilities. As each probability p takes on values within $[0, 1]$, the corresponding range for the function $f(p)$ is $[-e^{-1}, 0]$. Thus, $f(p)$ has a finite dynamic range and is defined for all values of p. Several methods for calculating logarithmic functions in hardware have been reported [28], but of particular interest is the multiple lookup table (LUT)-based approach introduced by Castro-Pareja and Shekhar [5]. This approach minimizes the error in representing $f(p)$ for a given number and size of LUTs and, hence, is accurate and efficient. Following this approach, the reported design implements $f(p)$ using multiple LUT-based piecewise polynomial approximation.

2.2. Parameterized Architectural Design

Implementations of signal processing algorithms using microprocessor- or DSP-based approaches are characterized by a fixed datapath width. This width is determined by the hardwired datapath of the underlying processor architecture. Reconfigurable implementation based on FPGAs, in contrast, allows the size of datapath to be customized to achieve better tradeoffs among accuracy, area, and power. Moreover, this customization can also change at run time to accommodate varying design requirements. The use of such custom data representation for optimizing designs is one of the main strengths of reconfigurable computing [29]. It has been contended that the most efficient hardware implementation of an algorithm is the one that supports a variety of finite precision representations of different sizes for its internal variables [8]. In this spirit, many commercial and research efforts have employed parameterized design style for intellectual property (IP) cores [30–34]. This parameterization capability not only facilitates reuse of design cores but also allows them to be reconfigured to meet design requirements.

During the design of the aforementioned architecture, we adopted a similar design style that allows configuration of the wordlengths of the internal variables. Hardware design languages such as VHDL and Verilog natively support hierarchical parameterization of a design through use of *generics* and *parameters*, respectively. This design style takes advantage of these language features and is employed for the design of all the modules described earlier. We highlight the main features of this design style using illustrative examples. Consider a design module with two input variables that computes an output variable through arithmetic manipulation of the input variables. The wordlength of the input variables (denoted by IP1_WIDTH, IP2_WIDTH) and that of the output variable (denoted by OP_WIDTH) are the design parameters for this module. The module can then be parameterized for these design variables as illustrated in Figure 2(a).

In a pipelined implementation of an operation, a module may have multiple internal pipeline stages and corresponding intermediate variables. Wordlengths chosen for these intermediate variables can also impact the accuracy and hardware requirements of a design. In our implementation scheme, we do not employ any rounding or truncation for the intermediate variables, but deduce their wordlengths based on the wordlengths of the input operands and the arithmetic operation to be implemented. For example, multiplication of two 8-bit variables will, at the most, require a 16-bit-wide intermediate output variable. A parameterized implementation of this scenario is illustrated in Figure 2(c). Sometimes it is also necessary to instantiate a vendor-provided or a third-party IP core, such as a first in/first out (FIFO) module or an arithmetic unit, within a design module. In such cases, we simply pass the wordlength parameters down the design hierarchy to configure the IP core appropriately and thereby maintain the parameterized design style (see, e.g., Figure 2(b)).

When signals cross module boundaries, the output wordlength and format (position of the binary point) of the source module should match the input wordlength and format of the destination module. This is usually achieved through use of a rounding strategy and right- or left-shifting of the signals. Adopting "rounding toward the nearest" strategy to achieve wordlength matching is expected to introduce the smallest error but requires additional logic resources. In our design, we therefore implement truncation (or "rounding toward zero" strategy), while the signal shifting is achieved through zero padding. Both these operations are parameterized and take into account the wordlengths and the format at the module boundaries (see, e.g., Figure 2(c)). Thus, this parameterized design style enables the architecture to support multiple wordlength configurations for its internal variables.

3. Multiobjective Optimization

The aforementioned architecture is designed to accelerate the calculation of MI for performing medical image registration. We have demonstrated this architecture to be capable of offering execution performance superior to that of a software implementation [6]. The accuracy of MI calculation (and, by extension, that of image registration) offered by this implementation, however, is a function of the wordlengths chosen for the internal variables of the design. Similarly, these wordlengths also control the hardware implementation cost of the design. For medical applications, the ability of

```
--Declaration of a parameterized entity
entity Module1 is
 generic (
  IP1_WIDTH : INTEGER;
  IP2_WIDTH : INTEGER;
  OP_WIDTH : INTEGER);
 port (
  s1 : IN STD_LOGIC_VECTOR (IP1_WIDTH-1 DOWNTO 0);
  s2 : IN STD_LOGIC_VECTOR (IP2_WIDTH-1 DOWNTO 0);
  o1 : OUT STD_LOGIC_VECTOR (OP_WIDTH-1 DOWNTO 0));
end Module1;
```

(a)

```
--Instantiation of a vendor supplied
--IP-core, scfifo. The width of the
--FIFO is set to be equal to that of
--the signal to be buffered (o1).
fifo1 : scfifo
 generic map (
  LPM_NUMWORDS => (2**LOG2_DEPTH),
  LPM_WIDTH => OP_WIDTH,
  LPM_WIDTHU => LOG2_DEPTH)
 port map (
data => o1,...);
```

(b)

```
--Declaration of an intermediate variable with appropriate wordlength
signal i1 : STD_LOGIC_VECTOR (IP1_WIDTH+IP2_WIDTH-1 DOWNTO 0);
i1 <= s1 * s2; --Arithmetic operation (multiplication)

--Truncation of LSBs: Performed if (IP1_WIDTH+IP2_WIDTH) >= OP_WIDTH
o1 <= i1(IP1_WIDTH+IP2_WIDTH-1 DOWNTO IP1_WIDTH+IP2_WIDTH-OP_WIDTH);

--Signal-shifting: Performed if IP1_WIDTH+IP2_WIDTH) < OP_WIDTH
o1 <= i1 & CONV_STD_LOGIC_VECTOR(0,OP_WIDTH-(IP1_WIDTH+IP2_WIDTH));
```

(c)

Figure 2: Parameterized architectural design: (a) declaration of a parameterized entity; (b) an example instantiation of a vendor-supplied IP-core; (c) usage of parameterized internal variables and an example of truncation and signal shifting, performed at the module boundaries.

an implementation to achieve the desired level of accuracy is of paramount importance. It is, therefore, necessary to understand the tradeoff between accuracy and hardware implementation cost for a design and to identify wordlength configurations that provide effective tradeoffs between these conflicting criteria. This multiobjective optimization allows a designer to systematically maximize accuracy for a given hardware cost limitation (e.g., imposed by a target device) or minimize hardware resources to meet the accuracy requirements of a medical application.

Section 3.1 provides a formal definition of this problem, and Section 3.2 describes a framework developed for multiobjective optimization of FPGA-based medical image registration.

3.1. Problem Statement

Consider a system Q that is parameterized by N parameters n_i $(i = 1, 2, \ldots, N)$, where each parameter can take on a single value from a corresponding set of valid values (v_i). Let the design configuration space corresponding to this system be S, which is defined by a set consisting of all N-tuples generated by the Cartesian product of the sets v_i, $\forall i$:

$$S = v_1 \times v_2 \times v_3 \times \cdots \times v_N. \tag{5}$$

The size of this design configuration space is then equal to the cardinality of the set S or, in other words, the product of

the cardinalities of the sets v_i:

$$|S| = |v_1| \times |v_2| \times |v_3| \times \cdots \times |v_N|. \quad (6)$$

For most systems, not all configurations that belong to S may be valid or practical. We, therefore, define a subset $\mathfrak{I}$ ($\mathfrak{I} \subseteq S$), such that it contains all the feasible system configurations. Now, consider m objective functions $(f_1, f_2, \ldots, f_m)$ defined for system Q, such that each function associates a real value for every feasible configuration $c \in \mathfrak{I}$.

The problem of multiobjective optimization is then to find a set of solutions that simultaneously optimizes the m objective functions according to an appropriate criterion. The most commonly adopted notion of optimality in multiobjective optimization is that of Pareto optimality. According to this notion, a solution c^* is *Pareto optimal* if there does not exist another solution $c \in \mathfrak{I}$ such that $f_i(c) \leq f_i(c^*)$, for all i, and $f_j(c) < f_j(c^*)$, for at least one j. The solution c^* is also called a *nondominated* solution because no other solution dominates (or is superior to) solution c^* as per the Pareto-optimality criteria. The set of Pareto-optimal solutions, therefore, includes all nondominated solutions.

Given a multiobjective optimization problem and a heuristic technique for this problem that attempts to derive Pareto-optimal or near Pareto-optimal solutions, we refer to solutions derived by the heuristic as "Pareto-optimized" solutions.

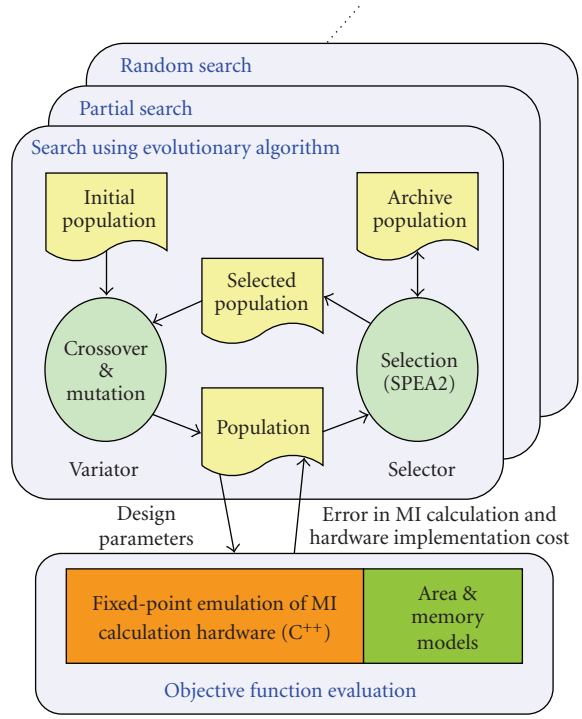


Figure 3: Framework for multiobjective optimization of FPGA-based image registration.

3.2. Multiobjective Optimization Framework

Figure 3 illustrates the framework that we have developed for multiobjective optimization of the aforementioned architecture. The framework has two basic components. The first is the search algorithm that explores the design space and generates feasible candidate solutions; the second is the objective function evaluation module that evaluates candidate solutions. The solutions and associated objective values are fed back to the search algorithm so that they can be used to refine the search. These two components are loosely coupled so that different search algorithms can be easily incorporated into the framework. Moreover, the objective function evaluation module is parallelized using a message passing interface (MPI) on a 32-processor cluster. With this parallel implementation, multiple solutions can be evaluated in parallel, thereby increasing search performance. These components are described in detail in the following sections.

3.2.1. Design Parameters

As described in Section 2.1, the architecture performs MI calculation using a fixed-point datapath. As a result, the accuracy of MI calculation depends on the precision (wordlength) offered by this datapath. The design parameters in this datapath define the design space and are identified and listed along with the corresponding design module (see Figure 1) in Table 1.

A fixed-point representation consists of an integer part and a fractional part. The numbers of bits assigned to these two parts are called the integer wordlength (IWL) and fractional wordlength (FWL), respectively. The individual numbers of bits allocated to these parts control the range and precision of the fixed-point representation. For this architecture, the IWL required for each design parameter can be deduced from the range information specific to the image registration application. For example, in order to support translations in the range of $[-64, 63]$ voxels, 7 bits of IWL (with 1 bit assigned as a sign bit) are required for the translation parameter. We used similar range information to choose the IWL for all the parameters, and these values are reported in Table 1. The precision required for each parameter, which is determined by its FWL, is not known a priori. We, therefore, determine this by performing multiobjective optimization using the FWL of each parameter as a design variable. In our experiments, we used the design range of [1, 32] bits for FWLs of all the parameters. The optimization framework can support different wordlength ranges for different parameters, which can be used to account for additional design constraints, such as, for example, certain kinds of constraints imposed by third-party IP.

The entropy calculation module is implemented using a multiple LUT—based approach and also employs fixed-point arithmetic. However, this module has already been optimized for accuracy and hardware resources, as described previously [5]. The optimization strategy employed in this earlier work uses an analytical approach that is specific to entropy calculation and is distinct from the strategy presented in this work. This module, therefore, does not participate in the multiobjective optimization framework of this paper, and we simply use the optimized configuration

Table 1: Design variables for FPGA-based architecture. Integer wordlengths are determined based on application-specific range information, and fractional wordlengths are used as parameters in the multiobjective optimization framework.

Architectural module	Design variable	Integer wordlength (IWL) (bits)	Fractional wordlength (FWL) range (bits)
Voxel coordinate transformation	Translation vector	7	[1, 32]
	Rotation matrix	4	[1, 32]
Partial volume interpolation	Floating image address	9	[1, 32]
Mutual histogram accumulation	Mutual histogram bin	25	[1, 32]

identified earlier. This further demonstrates the flexibility of our optimization framework to accommodate arbitrary designer- or externally optimized modules.

3.2.2. Search Algorithms

An exhaustive search that explores the entire design space is guaranteed to find all Pareto-optimal solutions. However, this search can lead to unreasonable execution time, especially when the objective function evaluation is computationally intensive. For example, with four design variables, each taking one of 32 possible values, the design space consists of 32^4 solutions. If the objective function evaluation takes 1 minute per trial (which is quite realistic for multiple MI calculation using large images), the exhaustive search will take 2 years. Even with the 32-processor cluster that we employed and assuming linear speedup, exhaustive search for a four-variable system will require about 3.5 weeks. This highlights the infeasibility of exhaustive search even for a system with a relatively small number of design variables. Consequently, we have considered alternative search methods, as described below.

The first method is *partial search*, which explores only a portion of the entire design space. For every design variable, the number of possible values it can take is reduced by half by choosing every alternate value. A complete search is then performed in this reduced search space. This method, although not exhaustive, can effectively sample the breadth of the design space. The second method is *random search*, which involves randomly generating a fixed number of feasible solutions. For both of these methods, Pareto-optimized solutions are identified from the set of solutions explored.

The third method is performing a search using evolutionary techniques. EAs have been shown to be effective in efficiently exploring large search spaces [18, 19]. In particular, we have employed SPEA2 [20], which is quite effective in sampling from along an entire Pareto-optimal front and distributing the solutions generated relatively evenly over the optimal tradeoff surface. Moreover, SPEA2 incorporates a fine-grained fitness assignment strategy and an enhanced archive truncation method, which further assist in finding Pareto-optimal solutions. The flow of operations in this search algorithm is shown in Figure 3.

For the EA-based search algorithm, the representation of the system configuration is mapped onto a "chromosome" whose "genes" define the wordlength parameters of the system. Each gene, corresponding to the wordlength of a design variable i, is represented using an integer *allele* that can take values from the set v_i, described earlier. Thus, every gene is confined to wordlength values that are predefined and feasible for a given design variable. The genetic operators for cross-over and mutation are also designed to adhere to this constraint and always produce values from set v_i, for a gene i within a chromosome. This representation scheme is both symmetric and repair-free and, hence, is favored by the schema theory [35] and is computationally efficient, as described by Kianzad and Bhattacharyya [36].

3.2.3. Objective Function Models and their Fidelity

Search for Pareto-optimized configurations requires evaluating candidate solutions and determining Pareto-dominance relationships between them. This can be achieved by calculating objective functions for all the candidate solutions and by relative ordering of the solutions with respect to the values of their corresponding objective functions. We consider the error in MI calculation and the hardware implementation cost to be the conflicting objectives that must be minimized for our FPGA implementation problem. We model the FPGA implementation cost using two components: the first is the amount of logic resources (number of LUTs) required by the design and the second is the internal memory consumed by the design. We treat these as independent objectives in order to explore the synergistic effects between these complementary resources. Because of the size of the design space and limitations resulting from execution time, it is not practical to synthesize and evaluate each solution. We, therefore, employ models for calculating objective functions to evaluate the solutions. The quality of the Pareto-optimized solutions will then depend on the fidelity of these objective function models.

The error in MI calculation can be computed by comparing the MI value reported by the limited-precision FPGA implementation against that calculated by a double-precision software implementation. For this purpose, we have utilized a bit-true emulator of the hardware. This emulator was developed in C++ and uses fixed-point arithmetic to accurately represent the behavior of the limited-precision hardware. It supports multiple wordlengths for internal variables and is capable of accurately calculating the MI value corresponding to any feasible configuration. We have verified its equivalence with the hardware implementation for a range of configurations and image transformations.

This emulator was used to compute the MI calculation error. The MI calculation error was averaged for three distinct image pairs (with different image modality combinations) and for 50 randomly generated image transformations. The same sets of image pairs and image transformations were used for evaluating all feasible configurations.

The memory required for a configuration is primarily needed for intermediate FIFOs, which are used to buffer internal variables, and the MH memory. For example, a 64-word-deep FIFO used to buffer a signal with a wordlength of b will require $64 \times b$ bits of memory. In our architecture, the depth of the FIFOs and the dimensions of the MH are constant, whereas their corresponding widths are determined by the wordlength of the design parameters. Using these insights, we have developed an architecture-specific analytical expression that accurately represents the cumulative amount of memory required for all internal FIFOs and MH. We used this expression to calculate the memory requirement of a configuration.

For estimating the area requirements of a configuration, we adopt the area models reported by Constantinides et al. [11, 37]. These are high-level models of common functional units such as adders, multipliers, and delays. These models are derived from the knowledge of the internal architecture of these components. Area cost for interconnects and routing is not taken into account in this analysis. These models have been verified for the Xilinx Virtex series of FPGAs and are equally applicable to alternative FPGA families and for application-specific integrated circuit (ASIC) implementations. These models have also been previously used in the context of wordlength optimization [11, 37, 38].

We further evaluated the fidelity [39] of these area models using a representative module, PV interpolator, from the aforementioned architecture. This module receives the fractional components of the FI address and computes corresponding interpolation weights. We varied the FWL of the FI address from 1 to 32 bits and synthesized the module using the Altera Stratix II and Xilinx Virtex 5 as target devices. For a meaningful comparison, the settings for the analysis, synthesis, and optimization algorithms (e.g., settings to favor area or speed) for the design tools (Altera Quartus II and Xilinx ISE) were chosen to be comparable. After complete synthesis, routing, and placement, we recorded the area (number of LUTs) consumed by the synthesized design. This process was automated by using the Tcl scripting feature provided by the design tools and through the parameterized design style described earlier. We then compared the consumed area against that predicted by the adopted area models for all FWL configurations. The results of this experiment are presented in Figure 4. These results indicate that the area estimates (number of LUTs) predicted by the model are comparable to those obtained through physical synthesis for both the target devices. For quantitative evaluation, the fidelity of the area models was calculated as follows:

$$\text{Fidelity} = \frac{2}{N(N-1)}\left(\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} F_{ij}\right), \tag{7}$$

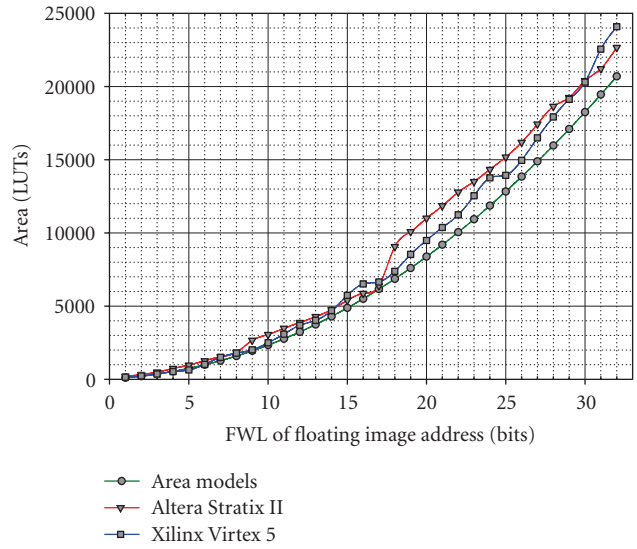


Figure 4: Comparison of the area values predicted by the adopted area models with those obtained after physical synthesis.

where

$$F_{ij} = \begin{cases} 1, & \text{if } \operatorname{sign}(S_i - S_j) = \operatorname{sign}(M_i - M_j), \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

In this equation, the M_is represent the values predicted by the area models; the S_is represent the values obtained after physical synthesis. The fidelity of the area models when evaluated with respect to the synthesis results obtained for both Altera and Xilinx devices was 1, which corresponds to maximum ("perfect") fidelity.

An interesting observation is that in some cases the high-level area models underestimate by as much as 25% the number of LUTs required. This can be explained by the fact that these models were calibrated using previous generation devices [11, 37]. It must be, however, noted that the Pareto-dominance relationship between the design configurations is maintained as long as the relative ordering (with respect to an objective function such as area) between two design configurations is preserved. Using more accurate area models will certainly improve the absolute prediction of area requirements corresponding to a given design configuration but, as such, will not affect the relative ordering of a set of design configurations. Designing accurate area models that take into account the latest devices, cross-vendor FPGA architectures, special-purpose computational units, and various synthesis optimizations is nevertheless important and will be a topic of a future investigation. The perfect fidelity we achieved for the current area models indicates that the relative ordering of FWL configurations with respect to their area requirements is consistent for the model and synthesized designs. These results further validate the applicability of using the aforementioned area models for multiobjective optimization.

Table 2: Number of solutions explored by search methods.

Search method	Number of solutions explored
Partial search	65 536
Random search	6000
EA-based search	6000

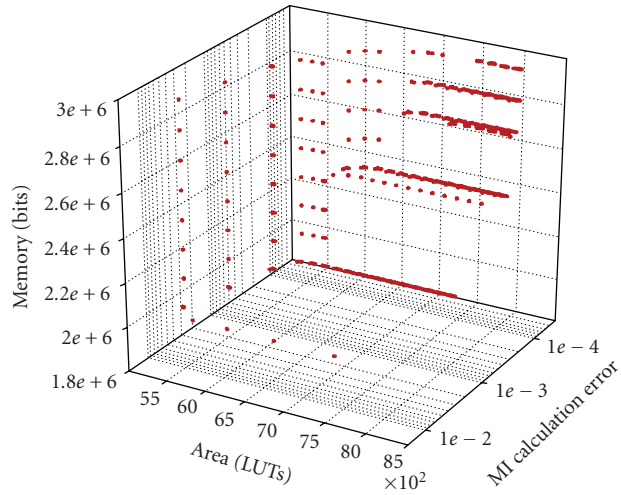
Table 3: Parameters used for EA-based search.

Parameter	Value
Population size	200
Number of generations	30
Cross-over probability	1.0
Mutation probability	0.06

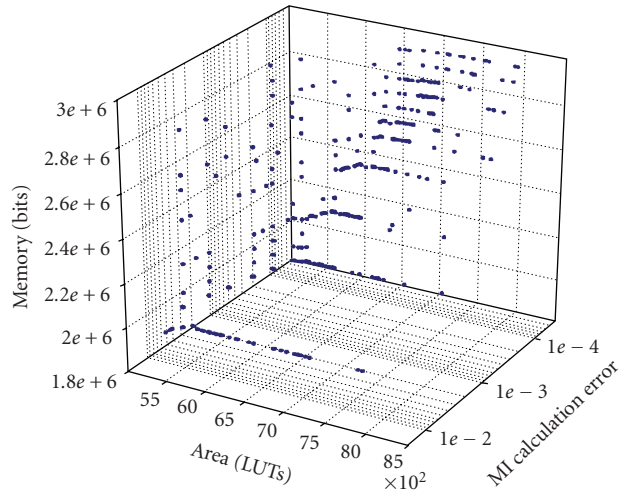
4. Results

We performed multiobjective optimization of the aforementioned architecture using the search algorithms outlined in Section 3. To account for the effects of random number generation, the EA-based search and random search were repeated five times each, and the average behavior from these repeated trials is reported. The number of solutions explored by each search algorithm in a single run is reported in Table 2. The execution time of each search algorithm was roughly proportional to the number of solutions explored, and the objective function evaluation for each solution took approximately 1 minute using a single computing node. As expected, the partial search algorithm explored the largest number of solutions. The parameters used for the EA-based search are listed in Table 3. These parameters were identified experimentally. For example, using a population size of 100 yielded similar search results; however, the diversity of the solutions found in the objective space was relatively poor. Similarly, increasing maximum number generations beyond 30 did not yield a significant improvement in the quality of the search solutions. The cross-over and mutation operators were chosen to be one-point cross-over and flip mutator, respectively. For a fair comparison, the number of solutions explored by the random search algorithm was set to be equal to that explored by the EA-based algorithm.

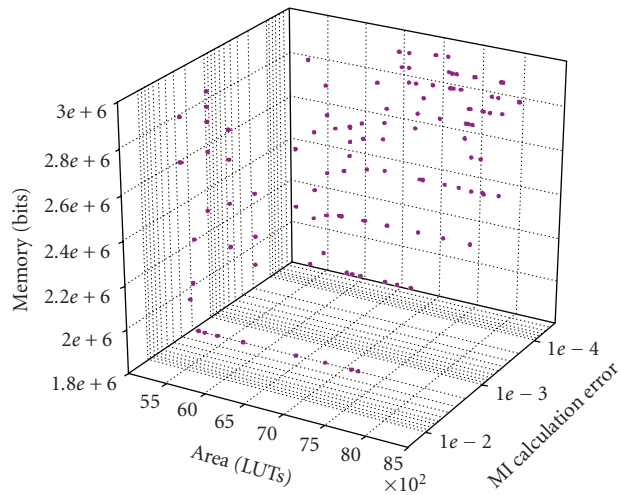
The solution sets obtained by each search method were then further reduced to corresponding nondominated solution sets using the concept of Pareto optimality. As described earlier, the objectives considered for this evaluation were the MI calculation error and the memory and area requirements of the solutions. Figure 5 shows the Pareto-optimized solution set obtained for each search method. Qualitatively, the Pareto front identified by the EA-based search is denser and more widely distributed and demonstrates better diversity than other search methods. Figure 6 compares the Pareto fronts obtained by partial search and EA-based search by overlaying them and illustrates that the EA-based search can identify better Pareto-optimized solutions, which indicates the superior quality of solutions obtained by this search method. Moreover, it must be noted that the execution time required for the EA-based search was more than 10 times faster than that required for the partial search.



(a) Partial search

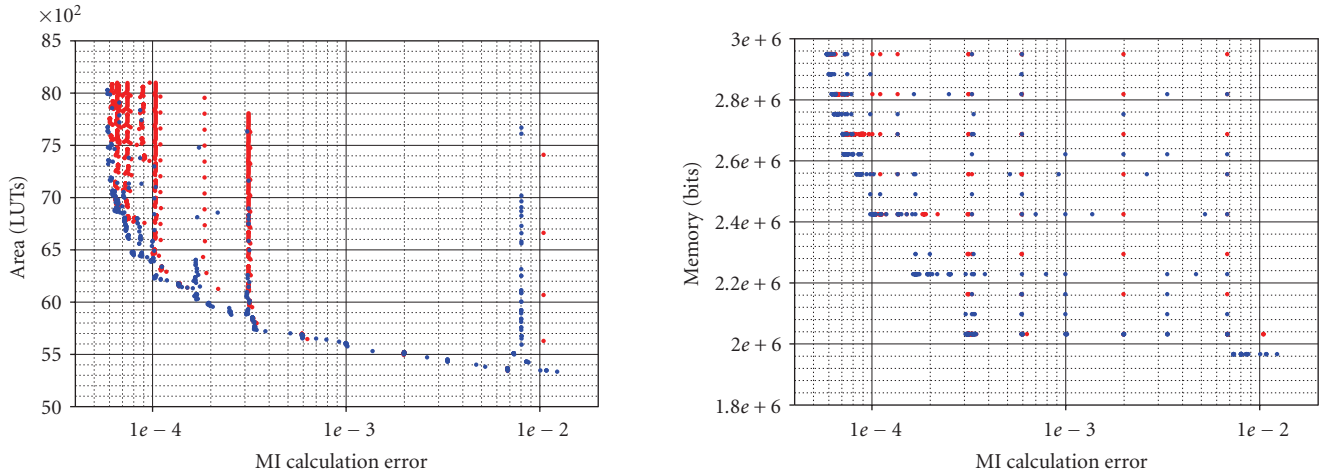


(b) EA-based search



(c) Random search

Figure 5: Pareto-optimized solutions identified by various search methods.



(a) Area versus MI calculation error

(b) Memory versus MI calculation error

Figure 6: Qualitative comparison of solutions found by partial search and EA-based search.

4.1. Metrics for Comparison of Pareto-Optimized Solution Sets

Quantitative comparison of the Pareto-optimized solution sets is essential in order to compare more precisely the effectiveness of various search methods. As with most real-world complex problems, the Pareto-optimal solution set is unknown for this application. We, therefore, employ the following two metrics to perform quantitative comparison between different solution sets. We use the ratio of non-dominated individuals (RNIs) to judge the quality of a given solution set, and the diversity of a solution set is measured using the cover rate. These performance measures are similar to those reported by Zitzler and Thiele [40] and are described below.

The RNI is a metric that measures how close a solution set is to the Pareto-optimal solution set. Consider two solution sets ($P1$ and $P2$) that each contain only nondominated solutions. Let the union of these two sets be P_U. Furthermore, let P_{ND} be a set of all nondominated solutions in P_U ($P_{\text{ND}} \subseteq P_U$). The RNI for the solution set P_i is then calculated as:

$$\text{RNI}_i = \frac{|P_i \bigcap P_{\text{ND}}|}{|P_{\text{ND}}|}, \tag{9}$$

where $|\cdot|$ is the cardinality of a set. The closer this ratio is to 100%, the more superior the solution set is and the closer it is to the Pareto-optimal front. We computed this metric for all the search algorithms previously described, and the results are presented in Figure 7. Our EA-based search offers better RNI and, hence, superior quality solutions to those achieved with either the partial or random search.

The cover rate estimates the spread and distribution (or diversity) of a solution set in the objective space. Consider the region between the minimum and maximum of an objective function as being divided into an arbitrary number of partitions. The cover rate is then calculated as the ratio of the number of partitions that is covered (i.e., there exists at least one solution with an objective value that falls within a given partition) by a solution set to the total number of partitions. The cover rate (C_k) of a solution set for an objective function (f_k) can then be calculated as:

$$C_k = \frac{N_k}{N}, \tag{10}$$

where N_k is the number of covered partitions and N is the total number of partitions. If there are multiple objective functions (e.g., m), then the net cover rate can be obtained by averaging the cover rates for each objective function as:

$$C = \frac{1}{m}\sum_{k=1}^{m} C_k. \tag{11}$$

The maximum cover rate is 1, and the minimum value is 0. The closer the cover rate of a solution set is to 1, the better coverage and more even (more diverse) distribution it has. Because the Pareto-optimal front is unknown for our targeted application, the minimum and maximum values for each objective function were selected from the solutions identified by all the search methods. We used 20 partitions/decades for MI calculation error (represented using a logarithmic scale), 1 partition for every 50 LUTs for the area requirement, and 1 partition for every 50 Kbits of memory requirement. The cover rate for all the search algorithms described earlier was calculated using the method outlined above and the results are illustrated in Figure 8. The EA-based search offers a better cover rate, which translates to better range and diversity of solutions when compared with either partial or random searches. In summary, our EA-based search outperforms the random search and is capable of offering more diverse and superior quality solutions when

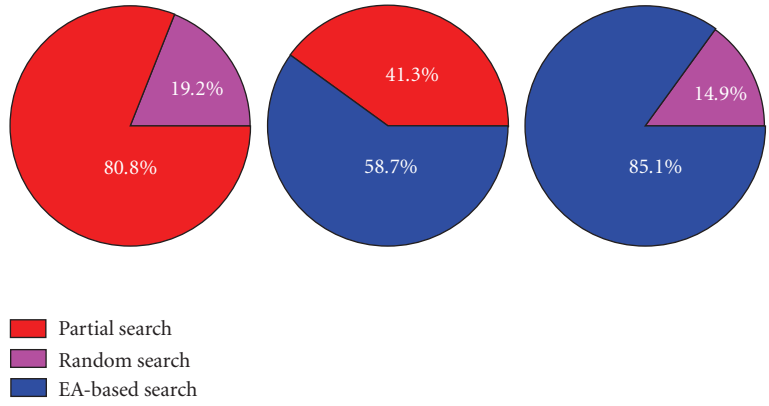


Figure 7: Comparison of search methods using the ratio of nondominated individuals (RNIs).

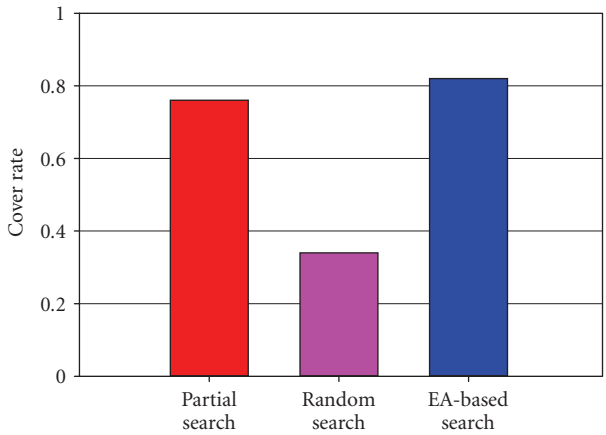


Figure 8: Comparison of search methods using cover rate.

compared with the partial search, using only 10% of the execution time.

4.2. Accuracy of Image Registration

An important performance measure for any image registration algorithm, especially in the context of medical imaging, is its accuracy. We did not choose registration accuracy as an objective function because of its dependence on data (image pairs), the degree of misalignment between images, and the behavior of the optimization algorithm that is used for image registration. These factors, along with its execution time, in our experience, may render registration accuracy as an unsuitable objective function, especially if there is nonmonotonic behavior with respect to the wordlength of design variables. Another important aspect is that the desired accuracy of registration depends on the application in which image registration is employed. For example, during an image-guided medical procedure high registration accuracy might be desired, whereas in a simple visualization task, slightly inaccurate image registration may be tolerated. Furthermore, in a multiresolution image registration approach slightly inaccurate (but, hardware resource-efficient) design configuration can be employed at the initial levels and a more accurate (but perhaps requiring more hardware resources) design configuration can be used at later levels. Thus, image registration accuracy is a constraint from an application perspective and, as such, is not used to guide the exploration of the design space. Instead, we used error in the MI calculation, which is relatively less application- and data-dependant, as an objective function.

Once the Pareto-optimized tradeoffs between MI calculation error and hardware resources are obtained through the presented approach, a system designer could evaluate the performance of these Pareto-optimized design configurations in the context of a specific target application. This can be done by using a set of sample image pairs acquired for that target application. To demonstrate the feasibility of this approach, we selected CT-CT registration as an example application. We randomly selected five clinical image pairs for this analysis and registered them using design configuration corresponding to each Pareto-optimized solution. These image pairs had the dimensions of $256 \times 256 \times 212$–$335$ voxels and the resolution of 1.4–$1.7\,\text{mm} \times 1.4$–$1.7\,\text{mm} \times 1.5\,\text{mm}$. This image registration was performed using the aforementioned bit-true simulator. The result of registration was then compared with that obtained using double-precision software implementation. Registration accuracy was calculated by comparing deformations at the vertices of a cuboid (with size equal to half the image dimensions) located at the center of the image. The results of this analysis, which establish the relationship between MI calculation error and the registration error specific to this application of CT-CT registration, are reported in Figure 9. It must be noted that each point in this plot represents a valid design configuration. As expected, there is a good correlation between the MI calculation error and the accuracy of image registration. This demonstrates that optimized tradeoff curves between MI calculation error and hardware cost, as identified by our reported analysis, can be used to represent the relationships between registration accuracy and hardware cost with high fidelity. These relationships can then be used to identify a design configuration in order to achieve desired registration accuracy for this example application of CT-CT registration. Similar relationships specific to a target application (e.g., PET-CT registration) can be generated using the aforementioned approach.

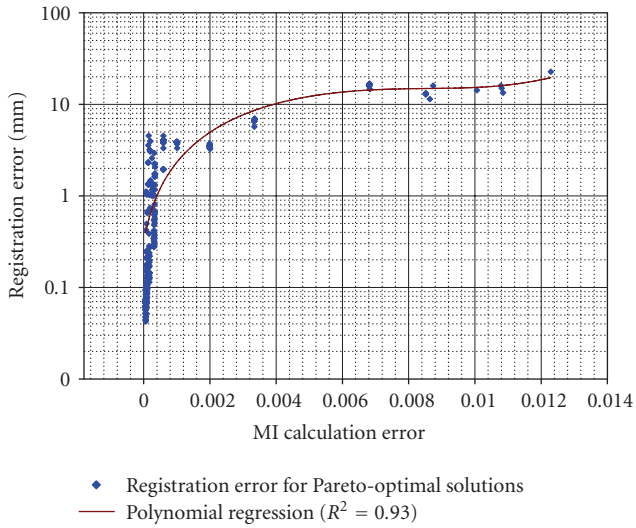


Figure 9: Relationship between MI calculation error and image registration error for an example application of CT-CT registration.

4.3. Postsynthesis Validation

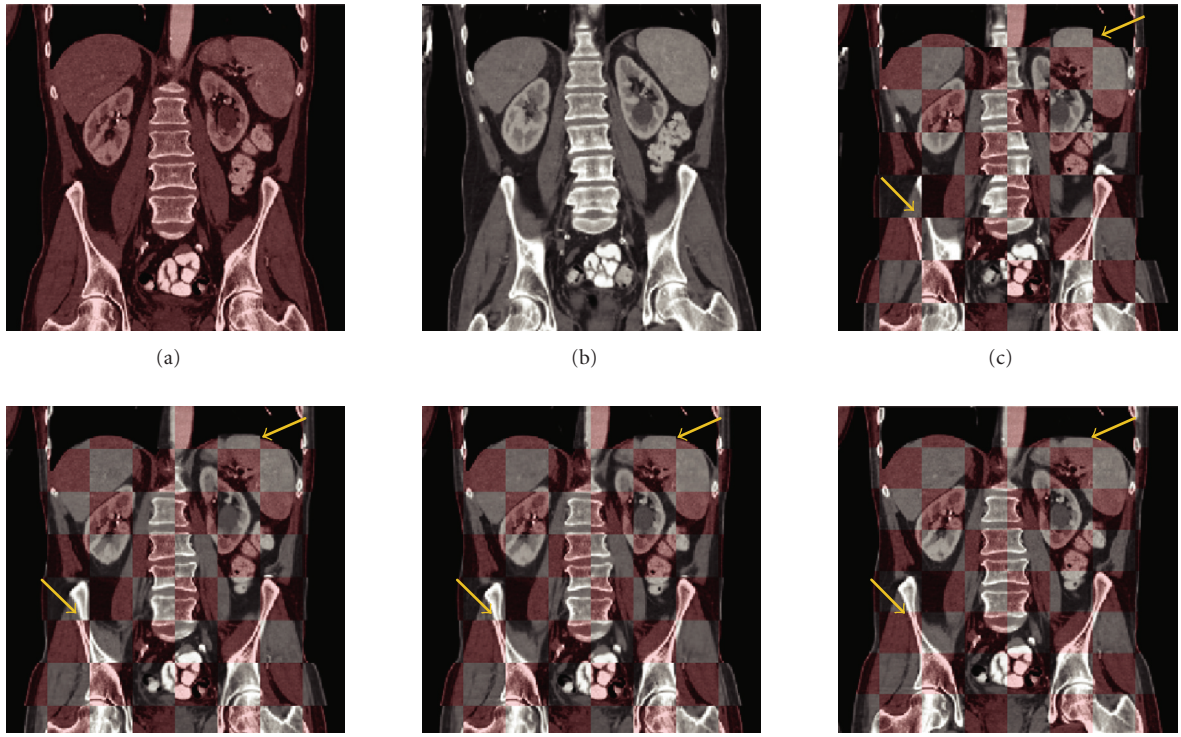
We performed further validation of the presented multiobjective optimization strategy through physical design synthesis. We identified three solutions from the Pareto-optimized set obtained using the EA-based search and synthesized the aforementioned architecture with configurations corresponding to these solutions. These solutions were identified with no specific clinical application in mind, but such that the tradeoff between various objective functions (MI calculation error, area, and memory) can be readily appreciated. Figure 9 reports the registration accuracy (calculated using the bit-true emulator that we developed) for all the Pareto-optimized design configurations. The system designer will have access to all the Pareto-optimized design configurations along with their expected MI-calculation error and hardware resource requirements, and, as such, can select a design configuration to meet the requirements of a given application.

These three configurations, which offer gradual tradeoff between hardware resource requirement and error in MI calculation, are listed in the first column of Table 4. The wordlengths associated with each configuration correspond to the FWLs of the design variables identified in Table 1. The design was synthesized for these configurations and the resulting realizations were implemented using an Altera Stratix II EP2S180F1508C4 FPGA (Altera Corporation, San Jose, Calif, USA) on a PCI prototyping board (DN7000K10PCI) manufactured by the Dini Group (La Jolla, Calif, USA). We then evaluated the performance of the synthesized designs and compared it with that predicted by the objective function models. The results of this analysis are summarized in Table 4 and are described below.

The error in MI calculation was computed by comparing the MI value reported by the limited-precision FPGA implementation against that calculated by a double-precision software implementation. The MI calculation error was averaged for three distinct image pairs and for 50 randomly generated image transformations for each pair. These image pairs and the associated transformations were identical to those employed in the objective function calculation. In this case, the average MI calculation error obtained by all the design configurations was identical to that predicted by the objective function model. This is expected because of the bit-true nature of the simulator used to predict the MI calculation error. We repeated this calculation with a different set of three image pairs and 50 randomly generated new transformations associated with each image pair. The MI calculation error corresponding to this setup is reported in the second column of Table 4. The small difference when compared with the error predicted by the models is explained by the different sets of images and transformations used. The area and memory requirements corresponding to each configuration after synthesis are reported in columns three and four of Table 4, respectively. For comparison, we have also included the values predicted by the corresponding objective function models in parenthesis. It must be noted that for all three configurations, the relative ordering based on Pareto-dominance relationships with respect to each objective function is identical for both postsynthesis and model-predicted values.

We also evaluated the accuracy of image registration performed using the implementation corresponding to each design configuration. For this analysis, we considered the same five CT image pairs described above. As reported earlier, these image pairs had dimensions of 256 × 256 × 212–335 voxels and resolution of 1.4–1.7 mm × 1.4–1.7 mm × 1.5 mm. The image registration results for one of those image pairs are illustrated in Figure 10. The result of registration between the remaining image pairs was also qualitatively similar. The registration error was calculated by comparing the obtained registration results with that obtained using double-precision software implementation. The mean and standard deviations of the registration error corresponding to each configuration are reported in Table 4. Good correlation is seen between the MI calculation error and the registration error, reinforcing the results presented in Section 4.2.

The performance of the resultant design configuration in terms of its raw clock rate is an important measure of the quality of a design. This clock rate directly affects the maximum voxel throughput that can be achieved by the design and, consequently, has an impact on the execution speed of image registration. The speed of a design configuration depends on, among other factors, the wordlengths of the design parameters. For example, performing arithmetic and memory operations using parameters with wider wordlengths may incur additional latency. As a result, design configurations employing design parameters with wider wordlengths may be slightly slower, although more accurate, than design configurations with shorter wordlengths. To provide some insights about this phenomenon, we recorded the maximum clock rate achieved by each of the design configurations we identified for synthesis. This represents the maximum postsynthesis frequency at which the design can operate and is reported in the last column of Table 4. These



(a) (b) (c)

(d) (e) (f)

Figure 10: Results of image registration performed using the high-speed, reconfigurable implementation: (a) and (b) two distinct poses; (c) fusion of (a) and (b) using a checkerboard pattern. The misalignment between images is evident at the edges of the squares within the checkerboard pattern; (d)–(f) fusion images after registration using the identified design configurations. These configurations offer progressively reduced image registration error (3.82 mm, 1.57 mm, and 0.45 mm, resp.) and result in correspondingly improved image alignment. The arrows indicate representative regions with misalignment that are better aligned after registration.

results indicate that the Pareto-optimized designs are not unreasonably slow and that their performance is comparable to that achieved (200 MHz) for a user-optimized design reported earlier [6].

This postsynthesis validation further demonstrates the efficacy of the presented optimization approach for reconfigurable implementation of image registration. It also further demonstrates how the approach enables a designer to systematically choose an efficient system configuration to meet the registration accuracy requirements for a reconfigurable implementation.

5. Discussion

With the need for real-time performance in signal processing applications, an increasing trend is to accelerate computationally intensive algorithms using custom hardware implementation. A critical step in going to a custom hardware implementation is converting floating-point implementations to fixed-point realizations for performance reasons. This conversion process is an inherently multidimensional problem because several conflicting objectives, such as area and error, must be simultaneously minimized. By systematically deriving efficient tradeoff configurations, one can not only reduce the design time [10] but can also enable automated design synthesis [41, 42]. Moreover, these tradeoff configurations allow designers to identify optimized, high-quality designs for reconfigurable computing applications. Our work presented in this paper develops a framework for optimizing tradeoff relations between hardware cost and image processing accuracy in the context of FPGA-based medical image registration.

Earlier approaches to optimizing wordlengths used analytical approaches for range and error estimations [11–15]. Some of these have used the error propagation method (e.g., see [14]), whereas others have employed models of worst-case error [12, 15]. Although these approaches are faster and do not require simulation, formulating analytical models for complex objective functions, such as MI, is difficult. Statistical approaches have also been employed for optimizing wordlengths [43, 44]. These methods employ range and error monitoring for identifying appropriate wordlengths. These techniques do not require range or error models. However, they often need long execution times and are less accurate in determining effective wordlengths.

Some published methods search for optimum wordlengths using error or cost sensitivity information. These approaches are based on search algorithms such as "Local," "Preplanned," and "Max-1" search [16, 45]. However, for a given design scenario, these methods are limited to finding a single-feasible solution, as opposed to a multiobjective tradeoff curve. In contrast, the techniques that we have

Table 4: Validation of the objective function models using postsynthesis results. The wordlengths in a design configuration correspond to the FWLs of the design variables identified earlier (see Table 1).

Design configuration	Objective functions postsynthesis value (predicted value)			Registration error (mean ± standard deviations, mm)	Design speed (f_{max}) (MHz)
	MI calculation error	Area (no. of LUTs)	Memory (Mbits)		
$\{5, 6, 4, 9\}$	2.4×10^{-3} (2.1×10^{-3})	6527 (5899)	2.23 (2.23)	3.82 ± 1.24	211
$\{8, 9, 7, 12\}$	5.3×10^{-4} (5.2×10^{-4})	7612 (6754)	2.45 (2.45)	1.57 ± 0.69	197
$\{9, 12, 10, 17\}$	7.7×10^{-5} (7.8×10^{-5})	10356 (8073)	2.81 (2.81)	0.45 ± 0.16	184

presented in this paper are capable of deriving efficient tradeoff curves across multiple objective functions.

Other heuristic techniques that take into account tradeoffs between hardware cost and implementation error and enable automatic conversion from floating-point to fixed-point representations are limited to software implementations only [42]. Also, some of the methods based on heuristics do not support different degrees of fractional precision for different internal variables [12]. In contrast, our framework allows multiple fractional precisions, supports a variety of search methods, and thereby captures more comprehensively the complexity of the underlying multiobjective optimization problem.

Other approaches to solve this multiobjective problem have employed weighted combinations of multiple objectives and have reduced the problem to mono-objective optimization [38]. This approach, however, is prone to finding suboptimal solutions when the search space is nonconvex [17]. Some methods have also attempted to model this problem as a sequence of multiple mono-objective optimizations [46]. The underlying assumption in this approximation, however, is that the design parameters are completely independent, which is rarely the case in complex systems. Modeling this problem as an integer linear programming formulation has also been shown to be effective [11]. But this approach is limited to cases in which the objective functions can be represented or approximated as linear functions of design variables.

EAs have been shown to be effective in solving various kinds of multiobjective optimization problems [18, 19] but have not been extensively applied to finding optimal wordlength configurations. One of the earlier attempts at using multiobjective EA formulation for wordlength optimization was reported by Istepanian and Whidborne [47]. This approach employed a simplistic model for hardware complexity and was limited to linear systems only. Leban and Tasic [48] also reported EA-based wordlength optimization of adaptive filters. However, this work was limited to mono-objective optimization only. More recently, Han et al. [49] reported EA-based multiobjective wordlength optimization for a filtering application. This work, however, considered only linear objective functions and lacked postsynthesis validation. In contrast, our work demonstrates the applicability of EA-based search for finding superior Pareto-optimized solutions in an efficient manner, even in the presence of a nonlinear objective function. Moreover, our optimization framework supports multiple search algorithms and objective function models and may be extended to a wide range of other signal processing applications. A preliminary version of the work presented in this article is published in [50]. This paper represents an enhanced and more thorough version of that work. New developments that we have incorporated into this paper include elaborating on the parameterized architectural design, evaluating the fidelity of the objective function models, and verifying the applicability of the proposed methodology through postsynthesis validation. In summary, this work has presented a framework that is capable of performing multiobjective wordlength optimization and identifying Pareto-optimized design configurations even in the context of nonlinear and complex objective functions. Through postsynthesis validation, this work has also demonstrated the feasibility of such a multiobjective optimization framework in the context of a representative image processing application, medical image registration.

6. Conclusion

One of the main strengths of reconfigurable computing over general-purpose processor-based implementations is its ability to utilize more streamlined representations for internal variables. This ability can often lead to superior performance and optimized fabric utilization in reconfigurable computing applications. Given this advantage, it is highly desirable to automate the derivation of optimized design configurations that can be switched among at run time. Toward that end, this paper has presented a framework for multiobjective wordlength optimization of finite-precision, reconfigurable implementations. This framework considers multiple conflicting objectives, such as hardware resource consumption and implementation accuracy, and systematically explores tradeoff relationships among the targeted objectives. Our work has also further demonstrated the applicability of EA-based techniques for efficiently identifying Pareto-optimized tradeoff relations in the presence of complex and nonlinear objective functions. The evaluation that we have performed in the context of FPGA-based medical image registration demonstrates that such an analysis can be used to enhance automated hardware design processes and efficiently identify a system configuration that meets given design constraints. Furthermore, the multiobjective optimization approach that we have presented is not application-specific and, with additional work, may be extended to a multitude of other signal processing applications.

Acknowledgments

This work was supported by the U.S. Department of Defense (TATRC) under Grant DAMD17-03-2-0001. The authors thank Dr. Nancy Knight for her help in editing and refining this manuscript. The authors also thank the journal's reviewers for their feedback and suggestions in improving this manuscript.

References

[1] C. R. Castro-Pareja, O. Dandekar, and R. Shekhar, "FPGA-based real-time anisotropic diffusion filtering of 3D ultrasound images," in *Real-Time Imaging IX*, vol. 5671 of *Proceedings of SPIE*, pp. 123–131, San Jose, Calif, USA, January 2005.

[2] O. Dandekar, C. R. Castro-Pareja, and R. Shekhar, "FPGA-based real-time 3D image preprocessing for image-guided medical interventions," *Journal of Real-Time Image Processing*, vol. 1, no. 4, pp. 285–301, 2007.

[3] S. Venugopal, C. R. Castro-Pareja, and O. Dandekar, "An FPGA-based 3D image processor with median and convolution filters for real-time applications," in *Real-Time Imaging IX*, vol. 5671 of *Proceedings of SPIE*, pp. 174–182, San Jose, Calif, USA, January 2005.

[4] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, "FAIR: a hardware architecture for real-time 3-D image registratio," *IEEE Transactions on Information Technology in Biomedicine*, vol. 7, no. 4, pp. 426–434, 2003.

[5] C. R. Castro-Pareja and R. Shekhar, "Hardware acceleration of mutual information-based 3D image registration," *Journal of Imaging Science and Technology*, vol. 49, no. 2, pp. 105–113, 2005.

[6] O. Dandekar and R. Shekhar, "FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 2, pp. 116–127, 2007.

[7] M. L. Silva and J. C. Ferreira, "Support for partial run-time reconfiguration of platform FPGAs," *Journal of Systems Architecture*, vol. 52, no. 12, pp. 709–726, 2006.

[8] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "The multiple wordlength paradigm," in *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '01)*, pp. 51–60, Rohnert Park, Calif, USA, April-May 2001.

[9] G. A. Constantinides and G. J. Woeginger, "The complexity of multiple wordlength assignment," *Applied Mathematics Letters*, vol. 15, no. 2, pp. 137–140, 2002.

[10] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: a fixed-point design and simulation environment," in *Proceedings of Design, Automation and Test in Europe (DATE '98)*, pp. 429–435, Paris, France, February 1998.

[11] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Wordlength optimization for linear digital signal processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1432–1442, 2003.

[12] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGAs," in *Proceedings of Design, Automation and Test in Europe (DATE '01)*, pp. 722–728, Munich, Germany, March 2001.

[13] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 1999.

[14] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 108–120, Vancouver, Canada, June 2000.

[15] S. A. Wadekar and A. C. Parker, "Accuracy sensitive word-length selection for algorithm optimization," in *Proceedings of the IEEE International Conference on Computer Design (ICCD '98)*, pp. 54–61, Austin, Tex, USA, October 1998.

[16] H. Choi and W. P. Burleson, "Search-based wordlength optimization for VLSI/DSP synthesis," in *Proceedings of the 7th IEEE International Workshop on VLSI Signal Processing*, pp. 198–207, La Jolla, Calif, USA, October 1994.

[17] I. Das and J. E. Dennis, "A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems," *Structural and Multidisciplinary Optimization*, vol. 14, no. 1, pp. 63–69, 1997.

[18] M. S. Bright and T. Arslan, "Synthesis of low-power DSP systems using a genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 1, pp. 27–40, 2001.

[19] C. L. Valenzuela and P. Y. Wang, "VLSI placement and area optimization using a genetic algorithm to breed normalized postfix expressions," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 390–401, 2002.

[20] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Proceedings of Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems (EUROGEN '01)*, pp. 95–100, Athens, Greece, September 2001.

[21] 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.

[22] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration," *Medical Image Analysis*, vol. 2, no. 1, pp. 1–36, 1998.

[23] D. L. G. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration," *Physics in Medicine & Biology*, vol. 46, no. 1, p. 1, 2001.

[24] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens, "Multimodality image registration by maximization of mutual information," *IEEE Transactions on Medical Imaging*, vol. 16, no. 2, pp. 187–198, 1997.

[25] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever, "Mutual-information-based registration of medical images: a survey," *IEEE Transactions on Medical Imaging*, vol. 22, no. 8, pp. 986–1004, 2003.

[26] V. Walimbe and R. Shekhar, "Automatic elastic image registration by interpolation of 3D rotations and translations from discrete rigid-body transformations," *Medical Image Analysis*, vol. 10, no. 6, pp. 899–914, 2006.

[27] M. Doggett and M. Meissner, "A memory addressing and access design for real time volume rendering," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '99)*, pp. 344–347, Orlando, Fla, USA, May-June 1999.

[28] D. M. Mandelbaum and S. G. Mandelbaum, "A fast, efficient parallel-acting method of generating functions defined by power series, including logarithm, exponential, and sine, cosine," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 1, pp. 33–45, 1996.

[29] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proceedings: Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005.

[30] Altera Corp., "Altera IP Megacore Library," http://www.altera.com/literature/lit-ip.jsp.

[31] W. Luk, S. Guo, N. Shirazi, and N. Zhuang, "A framework for developing parametrised FPGA libraries," in *Proceedings of the 6th International Workshop on Field-Programmable Logic Smart Applications, New Paradigms and Compilers (FPL '96)*, pp. 24–33, Darmstadt, Germany, September 1996.

[32] W. Luk and S. McKeever, "Pebble: a language for parametrised and reconfigurable hardware design," in *Proceedings of the 8th International Workshop on Field-Programmable Logic Smart Applications, New Paradigms and Compilers (FPL '98)*, pp. 1–9, Tallinn, Estonia, August 1998.

[33] Xilinx Inc., "Xilinx Core Generator," http://www.xilinx.com/ise/products/coregen_overview.pdf.

[34] J. Zhao, W. Chen, and S. Wei, "Parameterized IP core design," in *Proceedings of the 4th International Conference on ASIC*, pp. 744–747, Shanghai, China, October 2001.

[35] T. Back, U. Hammel, and H. P. Schwefel, "Evolutionary computation: comments on the history and current state," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 3–17, 1997.

[36] V. Kianzad and S. S. Bhattacharyya, "Efficient techniques for clustering and scheduling onto embedded multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 7, pp. 667–680, 2006.

[37] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimum wordlength allocation," in *Proceedings of 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 219–228, Napa, Calif, USA, April 2002.

[38] K. Han and B. L. Evans, "Optimum wordlength search using sensitivity information," *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 92849, 14 pages, 2006.

[39] N. K. Bambha and S. S. Bhattacharyya, "A joint power/performance optimization technique for multiprocessor systems using a period graph construct," in *Proceedings of International Symposium on System Synthesis (ISSS '00)*, pp. 91–97, Madrid, Spain, September 2000.

[40] 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. 257–271, 1999.

[41] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimum and heuristic synthesis of multiple word-length architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 1, pp. 39–57, 2005.

[42] K. Kum, J. Kang, and W. Sung, "AUTOSCALER for C: an optimizing floating-point to integer C program converter for fixed-point digital signal processors," *IEEE Transactions on Circuits and Systems II*, vol. 47, no. 9, pp. 840–848, 2000.

[43] S. Kim, K. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Transactions on Circuits and Systems II*, vol. 45, no. 11, pp. 1455–1464, 1998.

[44] K. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, pp. 921–930, 2001.

[45] M. A. Cantin, Y. Savaria, and P. Lavoie, "A comparison of automatic word length optimization procedures," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 612–615, Phoenix, Ariz, USA, May 2002.

[46] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for Pareto-optimal configurations in parameterized system-on-a-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 4, pp. 416–422, 2002.

[47] R. S. H. Istepanian and J. F. Whidborne, "Multi-objective design of finite word-length controller structures," in *Proceedings of the Congress on Evolutionary Computation (CEC '99)*, pp. 1–68, Washington, DC, USA, August 1999.

[48] M. Leban and J. F. Tasic, "Word-length optimization of LMS adaptive FIR filters," in *Proceedings of the 10th Mediterranean Electrotechnical Conference (MALECON '00)*, pp. 774–777, Lemesos, Cyprus, May 2000.

[49] K. Han, A. G. Olson, and B. L. Evans, "Automatic floating-point to fixed-point transformations," in *Proceedings of the 40th Asilomar Conference on Signals, Systems, and Computers (ACSSC '06)*, pp. 79–83, Pacific Grove, Calif, USA, October 2006.

[50] O. Dandekar, W. Plishker, S. S. Bhattacharyya, and R. Shekhar, "Multiobjective optimization of FPGA-based medical image registration," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '08)*, pp. 183–192, Stanford, Calif, USA, April 2008.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 901328, 10 pages
doi:10.1155/2008/901328

Research Article

Dynamic Hardware Development

Stephen Craven[1] and Peter Athanas[2]

[1] *Department of Electrical Engineering, The University of Tennessee at Chattanooga, Chattanooga, TN 37403, USA*
[2] *Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic and State University, Blacksburg, VA 24061, USA*

Correspondence should be addressed to Stephen Craven, stephen-craven@utc.edu

Received 31 March 2008; Accepted 12 August 2008

Recommended by Michael Hubner

Applications that leverage the dynamic partial reconfigurability of modern FPGAs are few, owing in large part to the lack of suitable tools and techniques to create them. While the trend in digital design is towards higher levels of design abstractions, forgoing hardware description languages in some cases for high-level languages, the development of a reconfigurable design requires developers to work at a low level and contend with many poorly documented architecture-specific aspects. This paper discusses the creation of a high-level development environment for reconfigurable designs that leverage an existing high-level synthesis tool to enable the design, simulation, and implementation of dynamically reconfigurable hardware solely from a specification written in C. Unlike previous attempts, this approach encompasses the entirety of design and implementation, enables self-re-configuration through an embedded controller, and inherently handles partial reconfiguration. Benchmarking numbers are provided, which validate the productivity enhancements this approach provides.

1. Introduction

Field-programmable gate arrays (FPGAs) are a class of integrated circuits that can be reprogrammed numerous times after manufacture to implement arbitrary digital circuits. While FPGAs always lag custom application-specific ICs (ASICs) in performance, the significantly reduced non-re-occurring engineering costs make FPGAs attractive for a variety of applications. However, with very few exceptions, an FPGA in a deployed design implements a single static design, behaving exactly as if it were a fixed-function ASIC.

The ability to reconfigure itself in a deployed product offers FPGAs a distinct advantage over ASICs. Whereas an ASIC must allocate area to implement every digital circuit the application requires, regardless of how infrequently it is actually exercised, an FPGA only need be sized large enough to support the circuits being active at any one time. The research community has demonstrated the benefits of swapping circuits in and out of an FPGA in such diverse applications as image detection [1], gene sequencing [2], video processing [3], network applications [4], and instruction set extension [5].

Vendor and tool support for the dynamic partial reconfiguration (PR) of an FPGA has suffered from severe limitations in the past. PR design flows were poorly supported and frequently broken. Device configuration architectures required that an entire configuration column be loaded just to change a single bit in the FPGA configuration. Self-reconfiguration, through an internal configuration access port (ICAP), was limited to high-end devices, raising the cost of PR designs.

Recently, the PR landscape has experienced a change, driven in part by the growing importance of software defined radio (SDR), with its dynamic creation of radio waveforms. As the throughput requirements of SDR are impossible to meet with a processor and the configurability to implement arbitrary waveforms is beyond the capabilities of ASICs, the PR abilities of FPGAs are finally gaining tool support [6, 7]. The newer device families feature a configuration architecture that is more granular, increasing the speed and flexibility of PR [8]. Furthermore, PR capabilities have been extended to low-cost device families [9].

In spite of these trends, much work remains before PR design becomes an accepted practice. To develop

a PR application using current tools, a designer must learn the intricacies of the target architecture and nuances of unfamiliar design flows. Lacking models and tools to abstract away the low-level specifics of each different architecture, every porting of a PR application to a different device requires that the design process start anew. Simulation of a PR design before implementation must be forgone, owing to a lack of simulator support, complicating verification and debugging.

Concurrent with the changes in the PR landscape has been a push towards electronic system-level (ESL) design. ESL design involves raising the level of abstraction that a designer sees from the register transfer level (RTL) to something higher than what traditional hardware description languages (HDLs) provide [10]. The research community has experimented with high-level languages (HLLs) to lift the abstraction level [11], and their results are paying off with a variety of commercial ESL tools now available [12]. Design specifications can now be captured in a multitude of formats from graphical [13] to C [14], and automatically converted to synthesizable HDL by commercial high-level synthesis (HLS) tools.

Recognizing the potential for HLS to drastically reduce the complexity of PR design, several researchers have described development environments utilizing some form of high-level design capture specifically tailored to PR design [15–17]. Notable limitations in these projects, though, hinder their ability to take advantage of recent trends in configurable computing. The reliance of many of these projects on an external host hinders development of embedded applications and ignores embedded processor capabilities of modern FPGAs. The use of outdated design entry techniques, such as JBits [18], shackles several projects to older architectures.

This paper describes a new approach to PR application development that leverages a commercial HLS tool, integrates embedded processors, and provides models of communication and reconfiguration. Previous publications have described the methodology [19] and the language extensions to an HLS toolset [20]. This paper focuses on the implementation and testing of the development flow, providing design and productivity results that validate this approach.

Section 2 provides an overview of previous attempts to raise the level of abstraction in PR design. An overview of the approach of this paper is presented in Section 3, with Section 4 detailing the implementation of applications and providing benchmarking results. Finally, conclusions are discussed in Section 5.

2. Background

To address the difficulties in applying traditional design methodologies to PR applications, several researchers have proposed or implemented new methodologies targeting the requirements of PR hardware.

Janus [16] was an early effort at a unified PR application development environment centered around Java. Software for the host PC was written in Java, while the hardware for the multi-FPGA system was created in the same environment from JHDL, a Java-based structural hardware description language. Janus was developed under the coprocessor paradigm where the FPGA is essentially a slave to an external host processor. Partial reconfiguration and dynamic scheduling are not supported.

The PaDReH framework [21] focuses solely on hardware development, defining an open development flow permitting multiple methods of design capture, simulation, and partitioning to be used. Partial bitstream generation occurs within the Xilinx modular design flow, which is the only fully specified step in the framework. Little is provided to the designer in terms of tools or abstractions.

Synthesis and partitioning for adaptive reconfigurable computing systems (SPARCSs) [22] start with a behavioral VHDL description of the application separated into tasks communicating through shared memory or direct connections. Temporal and spatial scheduling occurs across multiple FPGAs. A high-level synthesis tool converts the behavioral description to RTL that is then processed with traditional tools.

The Institute for Software Integrated Systems (ISIS) describes a prototype model-integrated design environment for dataflow applications [23]. ISIS focuses on constraint-driven development and verification from a model-based approach. Tools automatically apply user-specified constraints to prune the design space. The development environment targets board-level designs comprised of heterogeneous computing elements (FPGAs, DSPs, processors, etc.), limiting the utility for FPGA-centric applications.

Recent work from Imperial College London defines abstractions of low-level details with an HLL-based approach to PR application development [15]. A modified form of C (RT-C) captures the design behavior at a high level, including configuration control. The RT-C is then translated into Handel-C [24], a commercial C-to-gates synthesis tool. An implementation flow generates the required configuration files, with configuration management handled by a host processor. The implementation flow, however, is based on JBits and therefore is limited to older architectures. Also, a manual translation is required to go from the Handel-C-generated HDL to JBits, and the resulting design is shackled to a host processor.

Brigham Young University developed a JHDL-based reconfigurable computing application framework (RCAF) with the distinguishing feature that the framework, consisting of control, communication, and debugging aids, is deployed in the finished product [25]. The framework assumes a tight integration of the FPGA with a host processor running a controlling Java programme. This framework does little to facilitate the capture of configuration management or the incorporation of embedded processors.

The Caronte PR framework defines a high-level development environment targeting coprocessor applications [26]. Simulation of PR is possible via SystemC, with design entry via HDLs or Impulse C [27]. Caronte's use of Impulse C differs from the work presented in this paper in that Caronte merely uses Impulse C to produce HDL and not to capture the totality of the application including the configuration

control. The bus-based communication of Caronte limits its applicability to streaming applications.

In addition to the projects described above, several researchers have explored the problem without producing a prototype design environment. Eisenring and Platzner's PR framework [28] describes a tool-independent design and implementation methodology in generic terms. Berkley's Stream Computations Organized for Reconfigurable Execution (SCORE) project [29] proposes a new FPGA-like architecture leveraging hardware pages to permit location-independent reconfiguration. While promising, no hardware has been produced.

These previous projects, summarized in Table 1, are each limited in important ways. Most assume a model of external configuration control, mandating the use of a host processor. For embedded application, this requirement is generally prohibitive. Many do not enable the use of partial reconfiguration. It is also interesting to note that no project has been extended, by its authors or others, since its initial implementation. This is perhaps in part due to the tight coupling of many of these frameworks to a specific architecture or design capture tool.

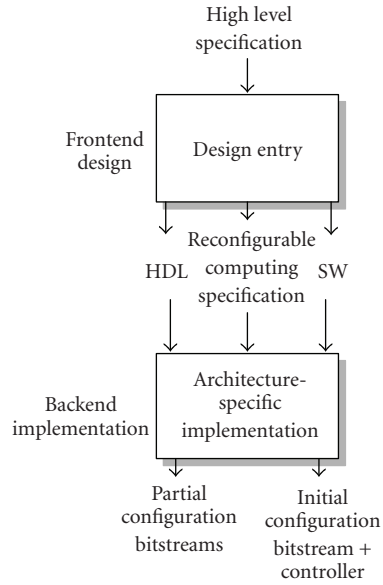


Figure 1: Combined design and implementation flow.

3. Approach

The goal of this project is to significantly reduce the effort required to deploy PR designs. To this end, a high-level development flow has been implemented that permits PR designs to be specified in C. Models of communication, computation, and reconfiguration have been defined that simplify design of streaming applications.

The development flow consists of a frontend, architecture-agnostic design flow, and a backend architecture-specific implementation flow. The design flow leverages an existing commercial HLS tool, modified to enable the capture and simulation of PR designs. By utilizing a commercial ESL tool, this work avoids the pitfalls of previous projects that relied heavily on outdated and unsupported tools such as JBits. The implementation flow is completely automated, encompassing floorplanning of the PR regions, insertion of a configuration controller, creation of the partial configuration bitstreams, and packaging of the configuration bitstreams for deployment.

Figure 1 presents the complete development flow, highlighting the exchange between the frontend and backend flows. To facilitate porting of designs to different architectures, the output of the frontend flow is completely architecture-agnostic. Due to variations in the configuration and clocking structure of different FPGA families, the backend flow may vary across architectures.

As conventional HDLs are not capable of capturing all aspects of PR designs, a reconfigurable computing specification format (RCSF) has been defined. The RCSF, expressed in XML, contains a list of reconfigurable modules, information concerning design connectivity, and the links to the HDL or SW that implements each module. A sample RCSF file is presented in Section 4.1. By editing this file, the designer can easily link to existing IP. A common use would be to replace a software test bench with the HDL that implements the actual interface to the application. Under this use model, a C model of the hardware IP could be leveraged to permit high-level simulation of the entire design early in the design cycle. This model would have to match the behavior of the hardware IP, but not the timing, as the high-level simulation is not cycle-accurate.

3.1. Abstractions

The models of computation and communication were selected to favor the traditional strengths of FPGAs, namely, streaming applications. Consisting of a repeatable schedule of computations operating on a steady flow of data, streaming applications are typically found in networking, signal processing, and cryptographic domains, all being strong suits of configurable logic. Multiple computational and communication models can accurately describe streaming applications, including several dataflow models and the communicating sequential processes (CSP) model [30]. In selecting an appropriate model, it was imperative that the actual functionality of hardware be captured and that commercial development tools support the model.

In CSP, an application is decomposed into a set of independently running processes, communicating only through unidirectional channels. Synchronization occurs during communication, with both the sender and receiver blocking until the transaction has completed. In contrast to some other dataflow paradigms, such as Kahn process networks [31] where communication occurs via infinitely deep FIFOs, CSP is directly implementable in hardware or software. Furthermore, tools and development environments exist supporting CSP design and implementation [14, 32, 33].

The implementation of an application using the CSP model of computation is straightforward. Communication channels can be created out of asynchronous FIFO buffers

Table 1: Previous PR development environments.

Project	Design entry	Model of computation	Architecture	Limitations
Janus	JHDL	Unspecified	Host + FPGA	No partial reconfiguration Requires host
SPARCS	Behavioral HDL	Dataflow	Host + FPGA	Requires macro library No partial reconfiguration
PaDReH	Multiple	Undefined	Standalone	Few defined tools
Model-Integrated	Dataflow graph	Dataflow	Independent	No partial reconfiguration Requires model library
RCAF	JHDL	Unspecified	Host + FPGA	No partial reconfiguration Requires host Few abstractions
Imperial College	RT-C	Dataflow	Limited by JBits	Requires host Manual translation
Caronte	Various	Coprocessor	Embedded proc	Limited automation

with minimal communication overhead. The FIFO-based communication permits easy integration with embedded processors as many Xilinx embedded processors feature fast simplex link (FSL) interfaces that are nothing more than asynchronous FIFO buffers linking the processor to peripherals [34].

To describe reconfiguration within a CSP model, the designer identifies a set of processes that are mutually exclusive in that only one of the set members is active in hardware at any one time. Figure 2 describes a cryptographic application where multiple decryption algorithms may be required, but never at the same time. Any process within the set of decryption cores may be selected for implementation, at which time the configuration manager reconfigures the FPGA to swap in the selected process. During reconfiguration, modules reading from or writing to the set undergoing reconfiguration will block until configuration is complete. This abstraction is similar to the swappable logic unit of Brebner [35] and the dynamic hardware modeling scheme of Luk [36].

This reconfiguration model enables the designer to utilize PR to extend an application breadth, by adding new functionality at runtime, or to extend an application depth, by swapping pipelined application stages in and out of the device. It is left to the designer to properly buffer results between the application stages.

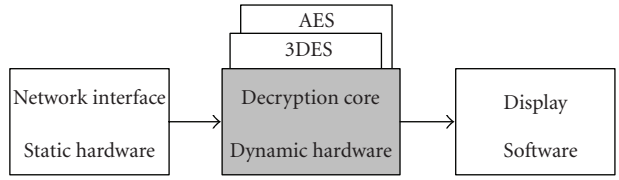


Figure 2: Set of mutually exclusive processes.

3.2. Frontend Design and Simulation

The language chosen for design entry is Impulse C, a commercial product of Impulse Accelerated Technologies, Inc. Impulse C [14] is an ANSI C-based language utilizing the same stream and process abstractions as Los Alamos National Lab's Streams-C work [11]. Based on the CSP model, Impulse C permits the application developer to describe hardware using a large subset of standard C. The CoDeveloper toolset performs high-level synthesis, translating Impulse C to synthesizable HDL.

Through an agreement with Impulse Accelerated Technologies, Inc., the CoDeveloper Impulse C application development environment has been obtained, along with the source code to the Impulse C simulation library. Modifications to the simulation library and corresponding extensions to the Impulse C language have been made permitting dynamic hardware to be simulated at a high level [20]. This modified language is referred to as DR Impulse C, highlighting its dynamic reconfiguration (DR) ability.

To describe PR applications in DR Impulse C, the programmer defines sets of mutually exclusive Impulse C processes. New Impulse C functions are utilized to create a set of reconfigurable processes and select a new dynamic process to execute in hardware. Applications described in DR Impulse C can be simulated by compiling the code in any C development environment. Each CSP process is spun off as a separate software thread communicating over shared buffers. PR is simulated by cleanly killing the executing thread and spinning off the new thread.

The frontend flow, shown in detail in Figure 3, consists of the CoDeveloper toolset for generating HDL from an Impulse C description, a preprocessor script for creating the RSCF file, and the GCC compiler for creating a simulation executable. Processes described in Impulse C can be marked

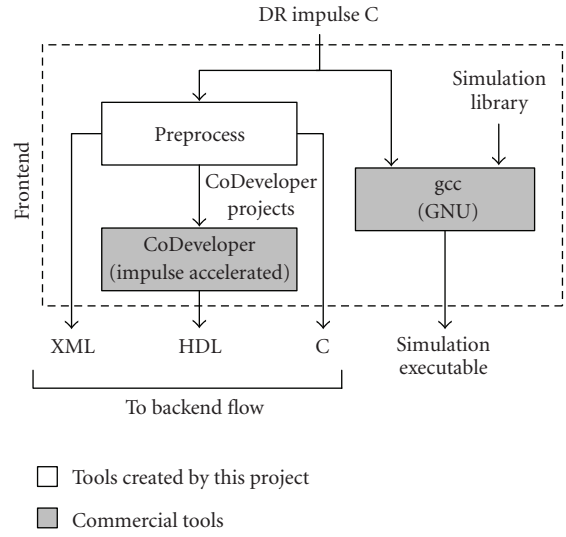


Figure 3: Frontend tool flow.

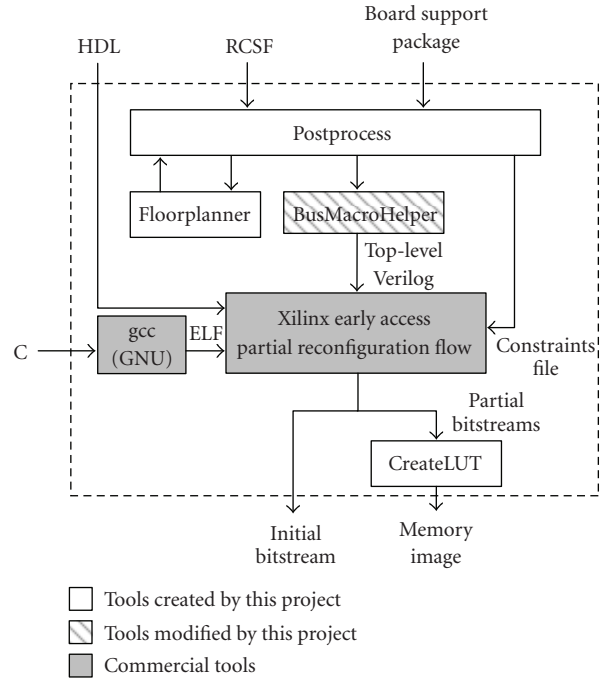


Figure 4: Backend tool flow.

for hardware implementation, in which the CoDeveloper tools convert the corresponding code to an HDL, or can be targeted to an embedded processor. The implementation flow handles the mapping of software processes to specific processors available on the target platform.

3.3. Backend Implementation

The architecture-specific implementation flow accepts the RCSF file, HDL modules, and C code from the frontend. In addition, a board support package (BSP) must be specified, supplying all the platform-specific information required to produce a deployable design. The implementation tool flow, shown in Figure 4, integrates tools automating placement, HDL generation, and clock creation.

The postprocess tool parses the RCSF and BSP, generating a top-level Verilog wrapper that instantiates each module in the design, along with the PR control modules, MicroBlaze controller, and clocking structure. The Floorplanner utility is responsible for creating area constraints for each reconfigurable region of the FPGA. This tool accepts as input a list of the resource requirements of each set and a list of keep-out regions. The keep-out regions correspond to areas of the FPGA that must be available for peripherals or soft processors, such as regions near critical I/Os. In keeping with other FPGA floorplanning projects [37–39], Floorplanner uses a simulated annealing algorithm to find a near optimal minimum of a cost function.

Unlike most previous works, Floorplanner is knowledgeable of the device configuration architecture, and attempts to find placements that minimize reconfiguration overhead. For the Xilinx Virtex-II and Virtex-II Pro architectures, where configuration frames run the entire height of the device, this involves finding a solution that has a high aspect ratio (height versus width) to use as much of the configuration frame as possible for the reconfigurable module. In the Virtex-4 architectures, where configuration frames are 16 CLBs tall, Floorplanner places all modules on configuration frame edges.

Floorplanner starts by first populating a list of module placements, called realizations. All possible realizations are considered in the creation of this list, with placements that are overly wasteful of resources being removed. Once a list of acceptable placements has been created, simulated annealing is performed to minimize the cost function:

$$cost = 10,000 * overlap + 10 * aspectError + waste + distance. \quad (1)$$

Module overlap, contained in *overlap* as the sum of all overlapping CLBs, is weighted orders of magnitude higher in the cost function to ensure that no two PR regions will overlap. *aspectError* penalizes the placements for having a poor aspect ratio with the ideal aspect ratio being dependent on the architecture. Higher ideal aspect ratios are used for the Virtex-II families to minimize reconfiguration overhead. *waste* is a measure of extra resources within the placement that will not be utilized on the device. The *distance* variable represents the total distance between reconfigurable regions, and it is used to minimize routing delays between reconfigurable regions.

Producing partial configuration bitstreams currently requires an Xilinx-supplied patch to the standard Xilinx ISE toolset. Among other changes, this patch constrains the router to keep routes inside a reconfigurable region. These modified tools make up the Xilinx early access PR (EAPR) flow. The EAPR flow requires that special connection points, called bus macros, surround reconfigurable modules, providing a stable connection point to the static hardware.

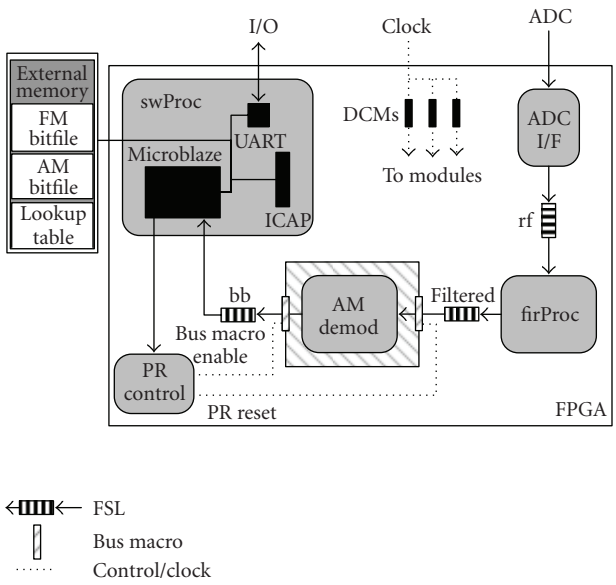


Figure 5: Final design implementation.

BusMacroHelper is a tool created for a related project that automatically inserts and places bus macros.

The CreateLUT tool creates a binary look-up table (LUT) that lists the size and location in memory of each partial bitstream enabling the configuration controller to find the desired partial bitstream. Additionally, the script concatenates the LUT and the partial bitstreams together into a single memory image to facilitate the automated download of the application to an FPGA.

Figure 5 presents an example implementation of a simple SDR application that may switch demodulation schemes. Several important aspects of this project are evident in the figure. The PR module *AM Demod* has been area-constrained to a specific location of the FPGA by the Floorplanner tool. All non-re-configurable modules are unconstrained, permitting the Xilinx tools to choose their optimum locations. All nonclock signals crossing the boundary between the static and PR regions must pass through a bus macro. As reconfiguration leaves the logic internal to a PR region in an undefined state, to stop the internal logic from producing random outputs that affect the rest of the system, the bus macro on the output of a PR region can be disabled. The tool flow automatically creates a PR control module for each PR region that disables the bus macros before reconfiguration and places any newly reconfigured module into a known good state by toggling the module reset line. Control of partial reconfiguration is handled by a MicroBlaze-based system running the user control code.

The CSP model permits each process to run at its own speed. To replicate this in hardware, each process receives its own clock, subject to resource availability. The FSL connections between processes are implemented as asynchronous FIFOs to enable cross-clock domain communication. The clocking structure is automatically generated using timing estimates from the synthesis tool.

4. Results

A video processing application, representative of streaming applications that benefit from PR, is described in this section followed by a comparison of the results obtained with this development flow and the results obtained manually following the Xilinx EAPR flow [40].

4.1. Application development

A video processing demonstration has been implemented using this development flow in which a video stream is filtered in real time with one of several filters. A separate filter acts on each of the three colors (red, green, and blue) and each can be independently reconfigured to implement an edge detector, a median image filter, or a pass-through. The edge detector and median image filter operate on a 5×5 window of pixels. The application forgoes a full frame buffer, using a separate columns process to buffer five lines of pixels, presenting a column of five pixels to the filters.

The filters and control logic are all described in DR Impulse C. For high-level simulation, separate test processes are defined that load an input image from a Windows Bitmap (BMP) file and translate filters' outputs into a BMP, as shown in Figure 6. The filtered output images in Figure 6 were produced by this Impulse C simulation.

Before implementation, the application RCSF is edited to replace these Impulse C test benches with the interface logic for the video card and video DAC, which are a part of the BSP of the Xilinx Virtex-II Pro XUP development board. This edit involves the modification of only eight lines of XML code. The original RCSF file is shown in Figure 7. Each CSP process is linked to an implementation folder containing the HDL description. Connectivity is expressed by associating each port to a stream.

The implemented design (the layout of which is seen in Figure 8) encompasses 63% of an Xilinx xc2vp30. The filters operate at 57 MHz, sufficiently fast to support the incoming 640×480 video stream at 60 Hz. If implemented as a static design, the hardware would have to include nine separate filters, that is, three filters for each of the three colors. The total area required by all nine filters would be 1707 slices. Partial reconfiguration reduces the area requirements to three instances of the largest filter, consuming 1328 slices across three reconfigurable regions, thus resulting in an area saving of 379 slices due to using PR. Any additional filters added to the system would increase this area saving.

4.2. Benchmarks

To quantify the advantages and disadvantages of the high-level development environment, a set of applications was implemented in this environment and compared to implementations made following the Xilinx EAPR flow. To more accurately simulate real-world design practices, the Xilinx EAPR flow was scripted following the PR documentation [40]. All designs were created by an experienced hardware designer familiar with the Xilinx configuration architecture and EAPR flow. Note that the results presented below do

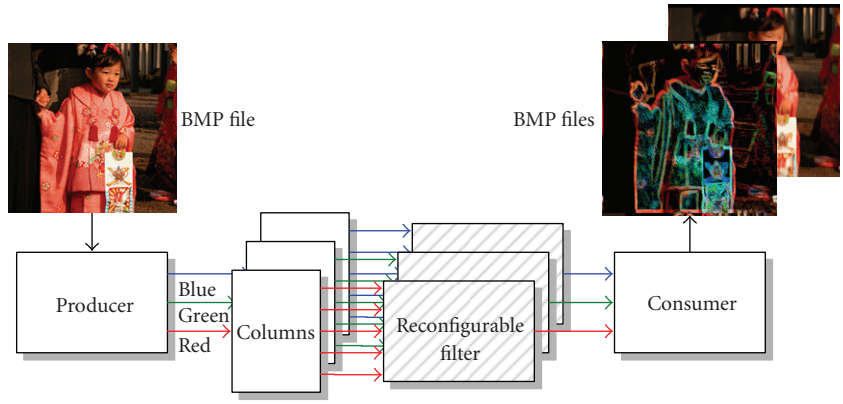


Figure 6: Video processing application.

```
<?xml version="1.0" ?>
<project implementation="xilinx_fsl" name="img_arch">
    <!-- Software testbenches for high-level simulation -->
    <process co_arch_reconfig="true" folder="sw_producerProcess"
        name="producerProcess" type="sw">
        <stream direction="output" name="blue_source_pixeldata" port_name="FSL0_M"/>
        <stream direction="output" name="green_source_pixeldata" port_name="FSL1_M"/>
        <stream direction="output" name="red_source_pixeldata" port_name="FSL2_M"/>
    </process>
    <process folder="sw_consumerProcess" name="consumerProcess" type="sw">
        <stream direction="input" name="blue_result_pixeldata" port_name="FSL0_S"/>
        <stream direction="input" name="green_result_pixeldata" port_name="FSL1_S"/>
        <stream direction="input" name="red_result_pixeldata" port_name="FSL2_S"/>
    </process>
        ⋮
</project>
```

Figure 7: RCSF file for simulated video processing design.

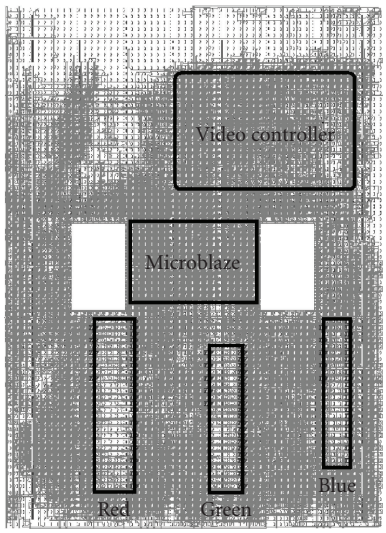


Figure 8: Floorplan of video processing implementation in an Xilinx xc2vp30.

not take into account the reduced skill set required by the high-level development environment. While some level of hardware experience is still required to create an application in DR Impulse C, it is significantly less than the low-level architecture-specific knowledge needed to follow the Xilinx EAPR flow.

The first application involved a reconfigurable coprocessor for an embedded MicroBlaze processor. This coprocessor, attached via an FSL interface, can be reconfigured to implement either a 32-bit integer divider or an integer square-root function. The descriptions for both functions were obtained from existing IP using the Xilinx Coregen tool and the OpenCores internet IP repository, in the case of the EAPR flow, and using example code provided with the Impulse C tools, in the case of this project's development environment.

The development time for both environments, from initial design description to working hardware implementation, was recorded. The PR region of the Xilinx EAPR flow was hand-placed, and it is 36% smaller than the Impulse C-based approach, owing to inefficiencies in HLS and automated floorplanning. Table 2 presents area and performance results at the module level. The Impulse C-generated divider compares well with the OpenCores divider, while the Coregen square-root function is significantly smaller than the Impulse C-generated module. The Impulse C-generated square-root function has a latency that is data-dependent. It should be noted that this high-level development environment can use existing IP and is not limited to Impulse C-created hardware though currently the implementation flow only supports IP with an FSL interface.

As presented in Table 3 for the integrated coprocessor application, the high-level development environment incurred a 71% penalty in average throughput and an 8% overall area penalty when compared to a manual implementation in the Xilinx EAPR flow. This throughput metric averages the best- and worst-case throughputs for the divider and square-root modules. The manual EAPR implementation ran the coprocessor at the system 100 MHz clock rate. The high-level development environment ran the coprocessor at 80% of the synthesis tool estimated clock rate for the slowest coprocess module. The performance penalty could be reduced by leveraging existing IP instead of using Impulse C-generated HDL. Additional gains are possible by dynamically modifying the clock rate of the coprocessor instead of running all coprocessors at the speed

Table 2: Coprocessor module performance benchmarks in an Xilinx xc2vp30.

Module	Area (slices/BRAMs/BMults)	Speed (MHz)	Throughput (ops/sec)
Divider (Impulse C)	258/0/0	134	3.8 (10^6)
Divider (OpenCores)	159/0/0	123	3.4 (10^6)
Square root (Impulse C)	760/1/9	56	0.7 (10^6)–4.7 (10^6)
Square root (CoreGen)	266/0/0	114	9.5 (10^6)

Table 3: Coprocessor application performance benchmarks.

Environment	Area (slices)	Average Throughput (ops/sec)
High level (Impulse C)	3118	1.6 (10^6)
Xilinx EAPR	2883	5.6 (10^6)

Table 4: Coprocessor application productivity benchmarks.

Environment	Frontend (h)	Backend (h)	Total (h)
High level (Impulse C)	0.8	5	5.8
Xilinx EAPR	6.2	7.4	13.6

of the slowest. The small area penalty is due to the superiority of hand-placed designs.

The high-level development approach netted a 57% reduction in overall development time, seen in Table 4. The frontend number indicates the time required to create the design description, whether in DR Impulse C or Verilog. The backend number represents the time required to take the design description through implementation, and includes any hardware debugging. While the DR Impulse C design bested the Verilog design for each metric, the majority of the productivity improvement came from the frontend design. Even with the EAPR flow leveraging existing IP, the time required to integrate this IP into a design was significantly greater than the time required to describe the application in Impulse C.

Cryptographic hash functions were used as a second benchmarking application. A reconfigurable region on the FPGA could be configured for either the MD5 or the SHA-1 standard. The hash functions were created from scratch using both Impulse C and Verilog. Area and performance numbers for each function are shown in Table 5. The Verilog-described SHA-1 consumed 12% more slices than the Impulse C design owing to the use of five independent memories to permit simultaneous access to the message data. This approach increases throughput at the expense of area. Had area been of primary concern, a Verilog design would have been smaller than the Impulse C-created hardware. The Impulse C MD5 and SHA-1 cores underperformed the Verilog cores by 39% and 63%, respectively.

Table 6 presents the performance results with the cryptographic modules integrating into the reconfiguration application. The high-level development environment imparts a 24% area penalty and a 48% performance penalty, compared to the conventional Verilog design.

Table 5: Cryptographic module performance benchmarks in an Xilinx xc2vp30.

Module	Area (slices/BRAMs)	Speed (MHz)	Throughput (blocks/sec)
MD5 (Impulse C)	1305/2	66	0.43 (10^6)
MD5 (Verilog)	613/0	61	0.71 (10^6)
SHA-1 (Impulse C)	1080/1	73	0.17 (10^6)
SHA-1 (Verilog)	1214/0	76	0.46 (10^6)

Table 6: Cryptographic application performance benchmarks.

Environment	Area (slices)	Throughput (blocks/sec)
High level (Impulse C)	2016	0.30 (10^6)
Xilinx EAPR	1632	0.58 (10^6)

Table 7: Cryptographic application productivity benchmarks.

Environment	Frontend (h)	Frontend (w/o MD5)	Backend (h)	Total (h)	Total (w/o MD5)
High level	8.1	1	3.3	11.3	4.3
Xilinx EAPR	6.3	2.2	6.3	12.5	8.5

The productivity advantage of the high-level development environment was hampered in this application by a bug in the Impulse C-generated hardware, as seen in Table 7. The time spent resolving this issue resulted in a 28% greater frontend design time for the high-level development environment than that for a Verilog-created design. If the MD5 design time was removed from consideration, the frontend design times for the high-level and conventional approaches are 1 and 2.2 hours, respectively. This 120% frontend design time improvement is more in line with the coprocessor productivity results. If the MD5 design and debug time are considered, the total development improvement of the high-level approach is 10%, while if the MD5 design time is excluded from both designs, the high-level productivity improvement increases to 49%, approximating the results for the coprocessor application.

While the performance and area results obtained from the HLS tool may limit its applicability to high-performance applications, this does not negate the utility of the presented dynamic hardware development environment. For designs with timing or area constraints that cannot easily be met

with current HLS tools, the user is free to leverage HDL from other sources. This project's design and implementation flows offer many benefits even in the case of hand-coded HDL. The design flow permits high-level simulation of the entire design from a simple C model of each module. The implementation flow automates the creation of placement and area constraints, a configuration controller, and partial bitstreams.

It should be noted that the performance and productivity results would likely improve under a model-based high-level design environment. While Impulse C is currently used for design capture, other development tools that support a dataflow model may be leveraged with only slight modifications to the simulation mechanism of the tools. One advantage of Impulse C is its ability to synthesize random control logic. However, for straight signal processing applications, graphical high-level design tools, such as the Xilinx system generator, may be more appropriate. The defined interface between this project's design and implementation flows facilitates the use of multiple design entry methods.

5. Conclusion

The introduction of HLS techniques into the design of partially reconfigurable hardware for FPGAs can significantly reduce development time. The observed reductions in development time of approximately 50% would likely be greater for larger designs and for designers not being intimately familiar with an FPGA low-level configuration architecture. The resulting performance penalty may be acceptable for a variety of applications given the development time improvements and the significantly reduced skill set required to implement reconfigurable applications. By leveraging high-level development techniques, the full potential of FPGAs can be made easily available to the designer.

References

[1] K.-N. Chia, H. J. Kim, S. Lansing, W. H. Mangione-Smith, and J. Villasenor, "High-performance automatic target recognition through data-specific VLSI," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 3, pp. 364–371, 1998.

[2] E. Lemoine and D. Merceron, "Run time reconfiguration of FPGA for scanning genomic databases," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '95)*, pp. 90–98, Napa Valley, Calif, USA, April 1995.

[3] D. Ross, O. Vellacott, and M. Turner, "An FPGA-based hardware accelerator for image processing," in *Proceedings of the International Workshop on Field Programmable Logic and Applications on More FPGAs (FPL '94)*, pp. 299–306, Oxford, UK, September 1994.

[4] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable network packet processing on the field programmable port extender (FPX)," in *Proceedings of the 9th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '01)*, pp. 87–93, Monterrey, Calif, USA, February 2001.

[5] M. J. Wirthlin and B. L. Hutchings, "Sequencing run-time reconfigured hardware with software," in *Proceedings of the 4th ACM International Symposium on Field Programmable Gate Arrays (FPGA '96)*, pp. 122–128, Monterey, Calif, USA, February 1996.

[6] J. Seely, "FPGA use in software-defined radios," EETimes, August 2004.

[7] "Xilinx, ISR offering SDR kit," EETimes, February 2006.

[8] Xilinx, Inc., "Virtex-4 configuration guide," 2007.

[9] Xilinx, Inc., "Spartan 3 generation configuration user guide," 2007.

[10] R. Goering, "High-level synthesis rollouts enable ESL," EETimes, May 2004.

[11] M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented FPGA computing in the Streams-C high level language," in *Proceedings of the 8th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '00)*, pp. 49–56, Napa Valley, Calif, USA, April 2000.

[12] B. Holland, M. Vacas, V. Aggarwal, R. DeVille, I. Troxel, and A. George, "Survey of C-based application mapping tools for reconfigurable computing," in *Proceedings of the 8th International Conference on Military and Aerospace Programmable Logic Devices (MAPLD '04)*, Washington, DC, USA, September 2005.

[13] Xilinx, Inc., "Xilinx System Generator for DSP version 8.2," user's guide, 2006.

[14] D. Pellerin and S. Thibault, *Practical FPGA Programming in C*, Prentice Hall, Upper Saddle River, NJ, USA, 2005.

[15] T. K. Lee, A. Derbyshire, W. Luk, and P. Y. K. Cheung, "High-level language extensions for run-time reconfigurable systems," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '03)*, pp. 144–151, Tokyo, Japan, December 2003.

[16] D. I. Lehn, R. D. Hudson, and P. M. Athanas, "Framework for architecture-independent run-time reconfigurable applications," in *Reconfigurable Technology: FPGAs for Computing and Applications II*, vol. 4212, pp. 162–172, Boston, Mass, USA, November 2000.

[17] P. Diniz, M. Hall, J. Park, B. So, and H. Ziegler, "Bridging the gap between compilation and synthesis in the DEFACTO system," in *Proceedings of the 14th International Workshop on Languages and Compilers for Parallel Computing (LCPC '01)*, vol. 2624, pp. 52–70, Cumberland Falls, KY, USA, August 2001.

[18] S. Guccione, D. Levi, and P. Sundararajan, "JBits: Java based interface for reconfigurable computing," in *Proceedings of the 2nd Annual Military and Aerospace Applications of Programmable Logic Devices Conference (MAPLD '99)*, pp. 1–9, Laurel, Md, USA, September 1999.

[19] S. Craven and P. Athanas, "A high-level development framework for run-time reconfigurable applications," in *Proceedings of the 9th Annual Conference on Military and Aerospace Programmable Logic Devices (MAPLD '06)*, Washington, DC, USA, September 2006.

[20] S. Craven and P. Athanas, "High-level specification of runtime reconfigurable designs," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '07)*, pp. 280–283, Las Vegas, Nev, USA, June 2007.

[21] E. Carvalho, N. Calazans, E. Brião, and F. Moraes, "PaDReH—a framework for the design and implementation of dynamically and partially reconfigurable systems," in *Proceedings of the 17th Symposium on Integrated Cicuits and Systems Design (SBCCI '04)*, pp. 10–15, Pernambuco, Brazil, September 2004.

[22] I. Ouaiss, S. Govindarajan, V. Srinivasan, M. Kaul, and R. Vemuri, "An integrated partitioning and synthesis system

for dynamically reconfigurable multi-FPGA architectures," in *Proceedings of the 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP '98)*, pp. 31–36, Orlando, Fla, USA, March-April 1998.

[23] T. Bapty, S. Neema, J. Scott, J. Sztipanovits, and S. Asaad, "Model-integrated tools for the design of dynamically reconfigurable systems," Tech. Rep., Institute for Software Integrated Systems, Vanderbilt University, Nashville, Tenn, USA, 2000.

[24] Celoxica, Inc., "Handel-C for hardware design," white paper, 2006.

[25] A. L. Slade, B. E. Nelson, and B. L. Hutchings, "Reconfigurable computing application frameworks," in *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '03)*, pp. 251–260, Napa, Calif, USA, April 2003.

[26] F. Ferrandi, M. D. Santambrogio, and D. Sciuto, "A design methodology for dynamic reconfiguration: the Caronte architecture," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, p. 163, Denver, Colo, USA, April 2005.

[27] A. Antola, M. D. Santambrogio, M. Fracassi, P. Gotti, and C. Sandionigi, "A novel hardware/software codesign methodology based on dynamic reconfiguration with impulse C and codeveloper," in *Proceedings of the 3rd Southern Conference on Programmable Logic (SPL '07)*, pp. 221–224, Mar del Plata, Argentina, February 2007.

[28] M. Eisenring and M. Platzner, "A framework for run-time reconfigurable systems," *The Journal of Supercomputing*, vol. 21, no. 2, pp. 145–159, 2002.

[29] E. Caspi, M. Chu, R. Huang, J. Yeh, J. Wawrzynek, and A. DeHon, "Stream computations organized for reconfigurable execution (SCORE)," in *Proceedings of the 10th International Conference on Field-Programmable Logic and Applications (FPL '00)*, pp. 605–614, Villach, Austria, August 2000.

[30] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[31] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–801, 1995.

[32] P. Ljung, "How to create fixed- and floating-point IIR filters for FPGAs," Programmable Logic Design Line, May 2006.

[33] A. Saifhashemi and P. A. Beerel, "High level modeling of channel-based asynchronous circuits using verilog," in *Proceedings of the Communicating Process Architectures Conference (CPA '05)*, vol. 63, pp. 275–288, Eindhoven, Netherlands, September 2005.

[34] J. A. Williams, N. W. Bergmann, and X. Xie, "FIFO communication models in operating systems for reconfigurable computing," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05)*, pp. 277–278, Napa, Calif, USA, April 2005.

[35] G. Brebner, "The swappable logic unit: a paradigm for virtual hardware," in *Proceedings of the 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '97)*, pp. 77–86, Napa Valley, Calif, USA, April 1997.

[36] W. Luk, N. Shirazi, and P. Y. K. Cheung, "Modelling and optimising run-time reconfigurable systems," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '96)*, pp. 167–176, Napa Valley, Calif, USA, April 1996.

[37] L. Cheng and M. D. F. Wong, "Floorplan design for multi-million gate FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 12, pp. 2795–2805, 2006.

[38] Y. Feng and D. P. Mehta, "Heterogeneous floorplanning for FPGAs," in *Proceedings of the 19th International Conference on VLSI Design Held Jointly with 5th International Conference on Embedded Systems Design (VLSID '06)*, pp. 257–262, Hyderabad, India, January 2006.

[39] L. Singhal and E. Bozorgzadeh, "Multi-layer floorplanning on a sequence of reconfigurable designs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 605–612, Madrid, Spain, August 2006.

[40] Xilinx, Inc., "Early access partial reconfiguration user guide," March 2006.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2008, Article ID 723950, 9 pages
doi:10.1155/2008/723950

Research Article

On the Use of Magnetic RAMs in Field-Programmable Gate Arrays

Y. Guillemenet,[1] L. Torres,[1] G. Sassatelli,[1] and N. Bruchon[2]

[1] *Le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), University of Montpellier II, UMR CNRS 5506, 161 rue ADA, 34392 Montpellier Cedex 5, France*
[2] *AREVA T&D/COGELEX, Branch Office, P.O. Box 87200, Riyadh 11642, Saudi Arabia*

Correspondence should be addressed to L. Torres, torres@lirmm.fr

Received 31 March 2008; Accepted 29 August 2008

Recommended by Michael Hubner

This paper describes the integration of field-induced magnetic switching (FIMS) and thermally assisted switching (TAS) magnetic random access memories in FPGA design. The nonvolatility of the latter is achieved through the use of magnetic tunneling junctions (MTJs) in the MRAM cell. A thermally assisted switching scheme helps to reduce power consumption during write operation in comparison to the writing scheme in the FIMS-MTJ device. Moreover, the nonvolatility of such a design based on either an FIMS or a TAS writing scheme should reduce both power consumption and configuration time required at each power up of the circuit in comparison to classical SRAM-based FPGAs. A real-time reconfigurable (RTR) micro-FPGA using FIMS-MRAM or TAS-MRAM allows dynamic reconfiguration mechanisms, while featuring simple design architecture.

1. Introduction

Most of the field-programmable gate arrays (FPGAs) are currently SRAM based [1]. In these devices, configuration memory is distributed throughout the chip. Each memory point has to be readable independently because each of these points is used to drive a transistor's gate or a Lookup table (LUT) input [1]. Nevertheless, for writing operation, the configuration memory is organized as a classical memory array. Speed limitation of the configuration is linked to the size of the words that the memory can write at a time. Multiplying the number of memory arrays can reduce this time and allows parallel loading of the configuration bit stream with partial dynamic reconfiguration capabilities. The small access time of the SRAM makes it popular in the FPGA industry. Nonetheless, its volatility and the need of an external nonvolatile memory to store the configuration data make it not suitable for nowadays embedded applications. Indeed, in embedded FPGA devices, the use of a nonvolatile internal memory like flash technology allows the chip to be powered down in the standby mode when not in use in order to reduce power consumption. Some FPGAs and CPLDs use flash memory like Actel's fusion products [2]. Indeed, these FPGAs use flash memory in their configuration layer which makes it ready to run at power up. However, distribution of the memory all over the chip raises some technological constraints and needs additional masks (10 to 15 for the flash technology) and dedicated process steps thereby increasing the chip cost. Moreover, these FPGAs are not sufficiently flexible because they do neither provide partial or dynamical reconfiguration nor fast reprogramming speed due to the high-access time inherent to the flash memory [3].

The use of nonvolatile memories such as MRAMs helps to overcome the drawbacks of classical SRAM-based FPGAs without significant speed penalty. Besides its advantage that lies in power saving during the standby mode, it also benefits to the configuration time reduction since there is no need to load the configuration data from an external nonvolatile memory as is usual in SRAM-based FPGAs. Furthermore, during the FPGA circuit operation, the magnetic tunneling junctions can be written which allows a dynamic (or shadowed) configuration and further increases the flexibility of FPGA circuits based on the MRAM.

On the other hand, MRAM memories have shown interesting features that include high-timing performance, high-density integration, reliable data storage, good endurance [4, 5], and low number of additional masks required for the magnetic postprocess. For this reason, several companies proposed commercial solutions, as, for instance, the Freescale's first standalone 256 K $\times$ 16 bits MRAM [6]

since 2006. In 2007, Honeywell, IBM, Infineon technologies, Freescale, NEC, Samsung, TSMC, Grandis, Renesas technologies, Crocus technologies, NVE are involved in R&D projects with this type of memory. Table 1 gives some comparisons between MRAM technology and classical flash and SRAM memories.

The first generation nonvolatile MRAM is the field-induced magnetic switching (FIMS). It uses the magnetic tunneling junction (MTJ) as a storage element and features better access time and endurance and less additional masks than the flash memory [7]. However, it requires high writing current and consequently very large transistors to generate it and a high write lines width, which penalizes the die area of circuits based on such a memory. The thermally assisted switching (TAS)-MRAM has shown improvements in reduction of the writing current and thus in the consumed power during write operation and even in selectivity when compared to the FIMS-MRAM [7, 8]. More advanced writing schemes in the MTJ, like the spin transfer torque (STT) [9] further allows reduction of the required writing current and the die area of the STT-MRAM and makes it promising for embedded applications.

In [10], authors have addressed the use of STT-MRAM in FPGAs circuits. Indeed, a writing circuit for an STT-MRAM and an STT-based nonvolatile register has been proposed in [10] and they have been both assessed competitive in terms of speed, power consumption, nonvolatility, and area in comparison with their counterparts implemented with more classical memories. The same logic circuits using the TAS writing scheme have been proposed more recently by [11].

We investigate here the use of both FIMS-MRAM and TAS-MRAM cells in FPGA circuits. This paper is organized as follows. In Section 2, we describe the remanent SRAM (RSRAM) cell principle and the structure of the read and write circuitry of the FIMS and TAS MRAMs. In Section 3, we describe the run-time reconfiguration. In Section 4, experimental results of an LUT-4 using emulated FIMS-MTJs are presented. Illustrative simulation results of the TAS-MRAM cell operation are shown in Section 5. Section 6 describes TAS-MRAM-based LUTs for a real-time reconfigurable approach. Finally, the last section describes the structure of a micro-FPGA using TAS-MRAM cells.

2. Magnetic RAM Cells

2.1. RSRAM Cell Principle

The structure we used to achieve a read operation is based on the Black Jr. and Das cell [12]. As shown in Figure 1, it consists in an unbalanced flip-flop (UFF), which is formed by a cross-coupled inverters and two resistances connected in series with the N-ch (or P-ch) MOSFETs of the inverter. These resistances must have two different values: $R_{\min}$ and $R_{\max}$. The complementary structure of the UFF requires the use of $R_{\min}$ ($R_{\max}$) in one branch and $R_{\max}$ ($R_{\min}$) in the other one. The transistor MN3 is used to control the read operation. The basic operation of this structure is explained as follows. When the transistor MN3 (which will be called hereafter the sensing transistor) is switched on (i.e., when the signal "Read" is at the high-logic level), the flip-flop goes into the metastable state where gates and drains of transistors in the cross-coupled inverters are at the same voltage, which nears $V_{dd}/2$. When the signal "Read" goes low, the transistor MN3 switches off and the unbalanced flip-flop goes into a stable state. Depending on the values of the resistances, one output of the UFF is pulled high (i.e., V_{dd}), while the second output is pulled down (i.e., 0 V). The two resistances depict a nonvolatile memory device (NVMD) such that magnetic tunneling junctions, of which equivalent resistance (magneto-resistance) can be switched from $R_{\min}$ ($R_{\max}$) into $R_{\max}$ ($R_{\min}$) in order to achieve two complementary bits at the outputs of the cross-coupled inverters. The sensed data is then stored in the flip flop and can be used as many times as needed. The structure particularity results in its dual storage facility: one magnetic nonvolatile stage (MTJs) and a CMOS volatile stage (CMOS latch). When a signal is applied on the read pin, the physical value (the magneto-resistance) is converted electrically (0 V or VDD) into the CMOS part. The dual storage facility allows new properties such that run-time and shadowed reconfiguration depicted in Section 3.

Transistors in the unbalanced flip-flop must be sized in such way to achieve a good stability of the cell operation. Moreover, the resistance values of the NVMDs must have a TMR (tunneling magneto-resistance) ratio (TMR = $(R_{\max} - R_{\min})/R_{\min}$) sufficiently high to ensure a good operation

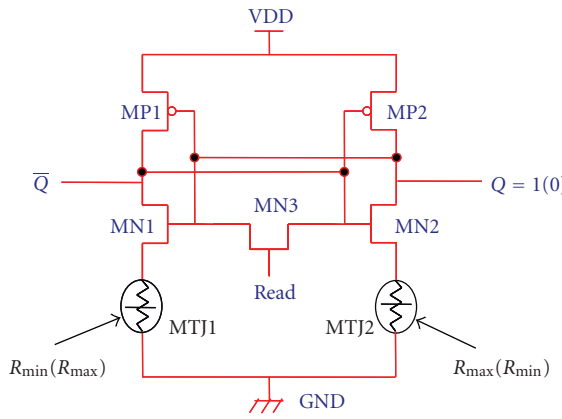


Figure 1: RSRAM cell based on the unbalanced flip-flop [12].

Table 1: Memories performance comparison.

	MRAM	SRAM	FLASH
Read speed	FAST	FASTEST	MEDIUM
Write speed	FAST	FASTEST	LOW
Cell area	LARGE	MEDIUM	LOW
Scalability	GOOD	GOOD	LIMITED
Endurence	MEDIUM	GOOD	LIMITED
Non-volatility	YES	NO	YES
Low voltage	YES	YES	NO
Rentension (years)	10	0	10
Number of additional mask	4	0	12
Radiation hardness	++	− −	− −

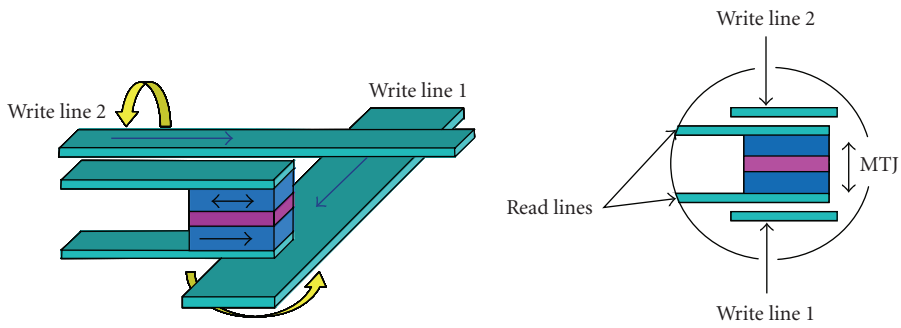


Figure 2: Write mechanism in the FIMS-MTJ.

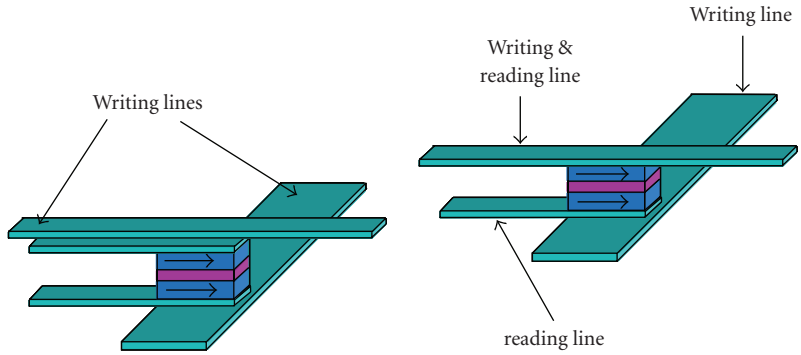


Figure 3: Suppression of the top reading line.

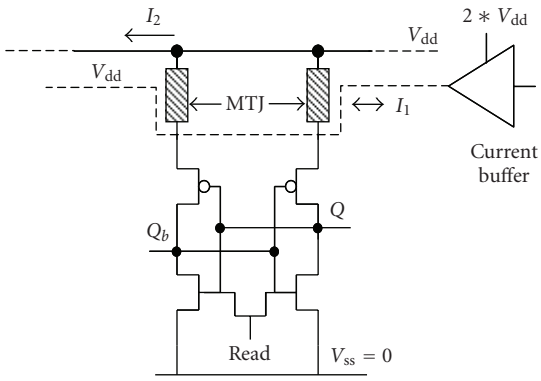


Figure 4: FIMS-MRAM structure.

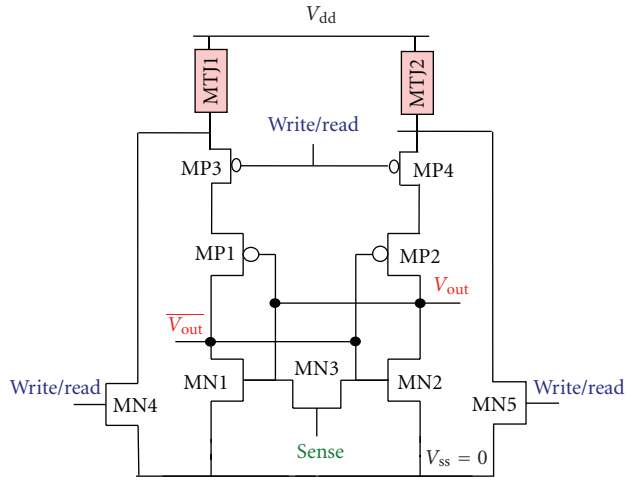


Figure 5: First read/write structure for a TAS-MRAM memory.

of the cell. For instance, simulations in 0.35 μm CMOS technology have shown that this TMR ratio must be at least 60%.

FIMS-MTJs and TAS-MTJs have been considered as NVMD devices in the RSRAM cell (see next sections).

2.2. Write Operation of the FIMS-MRAM

The FIMS-MTJ was the first concern of our previous studies [7, 13]. The FIMS magnetic tunneling junction is made of ferromagnetic layers separated by a very thin oxide one. Information is stored into the magnetic layers. Indeed, magnetic orientation of one of the layers is fixed once (pinned layer) and then used as a reference. The other layer (free layer) can be written thanks to two writing lines that are perpendicular to each other (as can be seen on Figure 2), the MTJ is sandwiched at the cross-point of these lines so that when a current is sent on both of the lines, magnetic fields generated around the lines will result in a field, which is high enough to change the magnetic orientation of the free layer. Relative magnetic orientation of these layers allows discriminating two different resistance values at the nodes of the junction (for parallel and antiparallel orientation in the magnetic layers). R_{min} corresponds to the resistance in

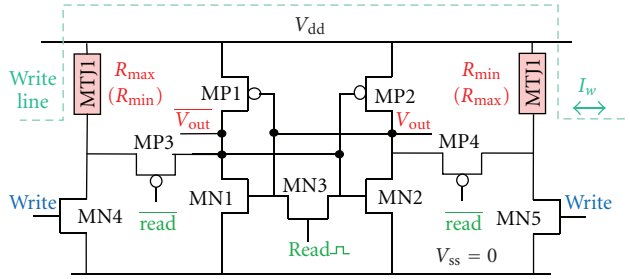


FIGURE 6: Second read/write structure for a TAS-MRAM memory.

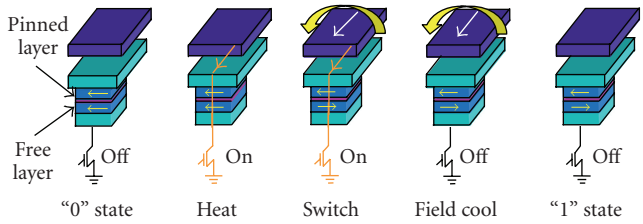


FIGURE 7: Step sequences of TAS write operation in an MTJ element.

the parallel mode (R_p), while R_{max} is the resistance in the antiparallel mode (R_{ap}).

2.3. Read Operation of the FIMS-MRAM

When using the FIMS approach to realize MRAM cells, the writing and the reading mechanisms are completely independent from each other, which explain the fact that the RSRAM structure (shown in Figure 1) does not need to be modified. The writing structure has to be added without affecting the stored data in the flip-flop structure. Nevertheless, the disadvantage of the FIMS-MTJ is the need of high-writing currents [7]. These currents can be reduced by slightly modifying the MTJ structure as shown in Figure 3. Indeed, the top writing and reading lines are merged so that the distance between the writing line and the free layer is reduced; lowering in the same time the current needed to write the free layer. Currently, the writing current depends mainly on the distance between the MTJ and the writing field line. During the reading phase, the read/write line has to be set to a potential that allows the reading of the junction. This solution implies that the reading and the writing mechanisms are not independent anymore from each other and the RSRAM structure has to be adapted to the MTJ one. The proposed solution is depicted in Figure 4; the top read/write line is connected to V_{dd}.

2.4. Read Operation of the TAS-MRAM

Two TAS-MRAM cells using the thermally assisted switching have been investigated and implemented in 0.35 μm in combination with TAS-MTJ (magnetic tunneling junction) postprocess. The structure of these two cells is depicted in Figures 5 and 6. Besides the unbalanced flip-flop and the transistor MN3 that enables the read operation, the

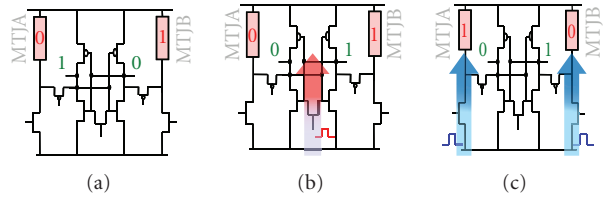


FIGURE 8: *Shadowed* reconfiguration mechanisms.

structure of the first cell (Figure 5) is composed of selection transistors MN4 and MN5 that are controlled on their gates by a signal "Write/Read," which enables the write operation. The same signal also drives the gates of transistors MP3 and MP4. These transistors act as "isolation transistors" in order to avoid any parasitic current on the UFF during write operation. The basic read operation of the first one (see Figure 5) is explained as follows. When the "Write/Read" signal is at the low-logic level, transistors MP3 and MP4 are switched on, while transistors MN4 and MN5 are switched off and disable any possible selectivity for a write operation. Hence the structure acts as a basic UFF and the written data in the MTJ devices is sensed when the signal "Sense" goes to the high-logic level and then stored in the flip-flop.

The structure of the second TAS-MRAM cell (depicted in Figure 6) consists in cross-coupled inverters, an NMOS transistor (MN3) to control the read operation, 2 MTJs for a nonvolatile storage, two transistors MP3 and MP4 driven on their gates by a signal "$\overline{read}$," which act as "isolation" transistors and similarly as in the first structure, selection transistors MN4 and MN5 are used to enable the write operation. This structure operates during the read mode as follows. When the signal "$\overline{read}$" is at low-logic level, transistors MP3 and MP4 are switched on. On the other hand, since the signal "write" is at the low-logic level, selection transistors MN4 and MN5 are switched off thereby disabling any write operation in the MTJs. The data written in the MTJs is then sensed because of the on-state of the transistor MN3, and latched in the flip-flop (read cycle is about 1 ns).

The first read/write structure of the TAS-MRAM memory (shown in Figure 5) needs only two control signals ("sense" and "Write/Read"), however, the higher number of stacked P-ch MOSFETs on each branch ((MP1, MP3) and (MP2, MP4)) compared to the second structure will lead to a slightly degraded access speed performance. On the other hand, the second structure needs an additional inverter gate in order to generate the control signal $\overline{read}$ from the signal "read."

2.5. Write Operation of the TAS-MRAM

Write operation follows the same step sequences in the two TAS-MRAM cells described above. When the selection transistor driven with a pulse signal on its gate is switched on, a heating current (~2 mA) is then generated through the magnetic tunneling junction. This current flow will subsequently heat the junction up to a blocking temperature

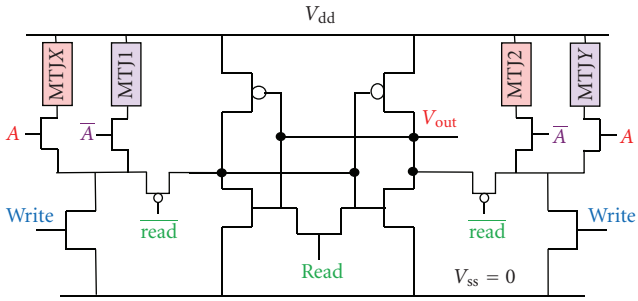


Figure 9: Multicontext TAS-MRAM cell.

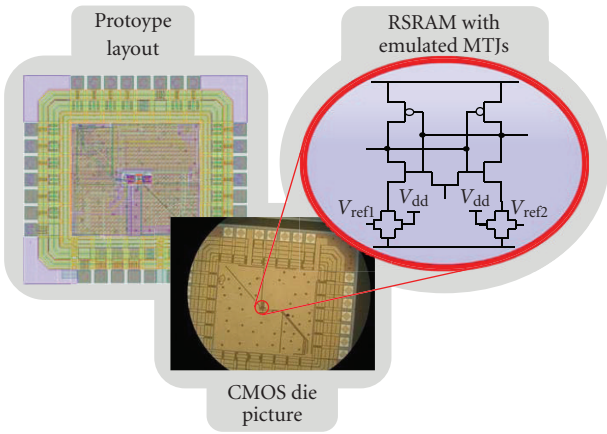


Figure 10: First test chip realized: layout (left), picture (center), emulated MTJ RSRAM cell schematic (right).

(~150°C) and simultaneously an external magnetic field which must be higher than the coercive is applied to switch the magnetic field in the free layer in parallel or antiparallel direction of that of the reference layer. The heat current pulse is then completed and the MTJ is cooled under the write magnetic field. To guarantee a good MTJ programming, the magnetic postprocess layout must respect specific design rules and each step of the writing sequences must respect a minimal duration. For the heat, switch, and field cool steps, the programming timing could be realized in less than 35 nanoseconds (for brief comparison SRAM: <10 nanoseconds, FLASH: >150 microseconds).

When the magnetic fields in the free layer and the reference layer are in parallel directions, then the magnetoresistance value is R_{min}. Reciprocally, it equals R_{max} when they are in antiparallel directions. Figure 7 illustrates the write structure and the different write step sequences in a TAS-MTJ element. Therefore, we can see the write-line on which a write current pulse is applied. The magnitude of this current must be high enough (at least 7 mA) such that the generated magnetic field is larger than the coercive magnetic field. The transistor shown in this figure depicts the selection transistor which is driven by a heating current pulse. The combination of a heating and a write magnetic field provides reliable write selectivity and prevents any addressing errors, for instance, in MRAM arrays.

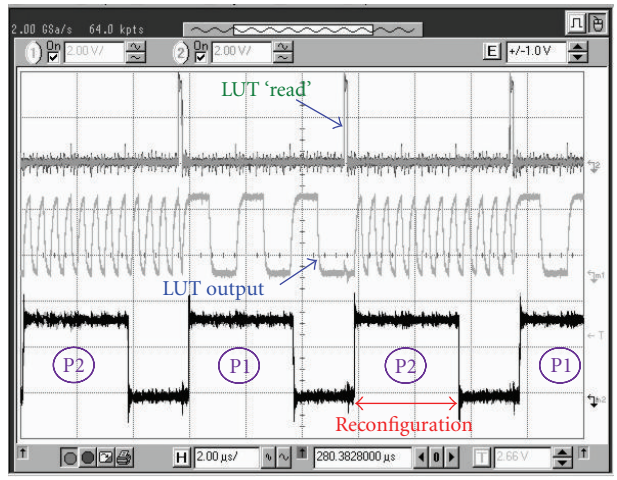


Figure 11: Experimental results illustrating the shadowed reconfiguration in our emulated MTJ RSRAM-based 4-inputs LUT. This RSRAM does not include any MRAMs.

3. Run-Time Reconfiguration

3.1. Shadowed Reconfiguration

The run-time reconfiguration is due to the redundancy of the information storage. Indeed, after a read cycle, information in the latch part and in the MRAM is the same. The run-time reconfiguration concerns the ability for the device to be in use while its configuration memory is rewritten; which is the case in circuits using latch. Figure 8 describes the different steps of this reconfiguration, from one configuration to another one.

In Figure 8(a), the circuit is using the data stored in the latch part which is different from the one stored in the TAS-MRAM. In Figure 8(b), a sense pulse is applied on the gate of the read transistor so that the value in the TAS-MRAM is transmitted to the circuit. During this step, the circuit cannot be in use. Then, in Figure 8(c), the circuit is running again with the new configuration and the TAS-MRAM can be written without disturbing the functioning of the circuit.

Thanks to the fast conversion time (transmission of the information from the MRAM to the latch), this structure allows a global FPGA or a part of it to be programmed in run-time since the new configuration is already distributed over the device and a single read sense signal on the gate of the read transistor is enough to have the new configuration in the CMOS circuitry.

3.2. Multicontext Reconfiguration

In a classical SRAM FPGA, an additional context is very consuming in terms of area since the configuration memory has to be duplicated. The advantage here comes from the fact that the MRAM store the configuration is laid over the CMOS. The structure we proposed to integrate multicontext in our circuits is the one depicted in Figure 9 which is derived from Figure 6.

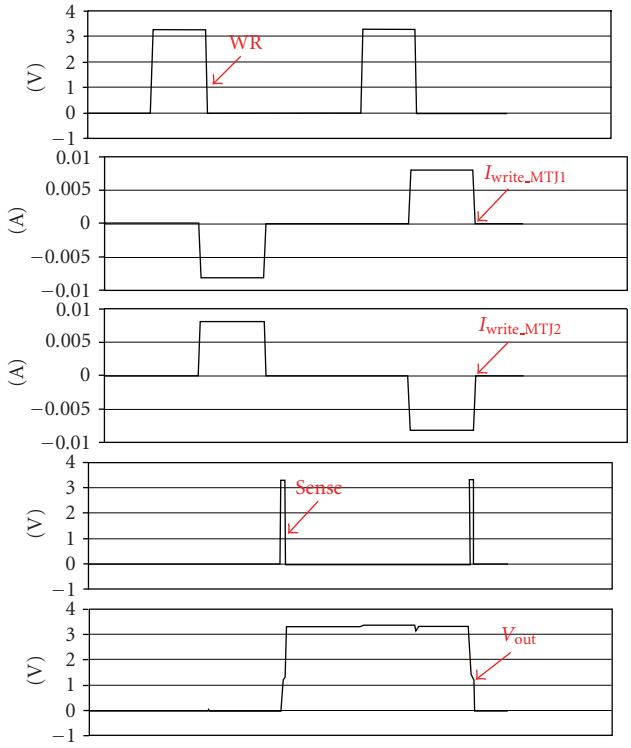


Figure 12: Simulation results obtained from the first structure of the TAS-MRAM in 0.35 μm CMOS combined with the TAS-MTJ technology, under a supply voltage V_{dd} = 3.3 V. The signal "WR" represents the signal called "write/read" as shown in the TAS-MRAM cell (see Figure 5). The signals I_{write_MTJ1} and I_{write_MTJ2} are the write current pulses applied to the MTJ1 and the MTJ2, respectively.

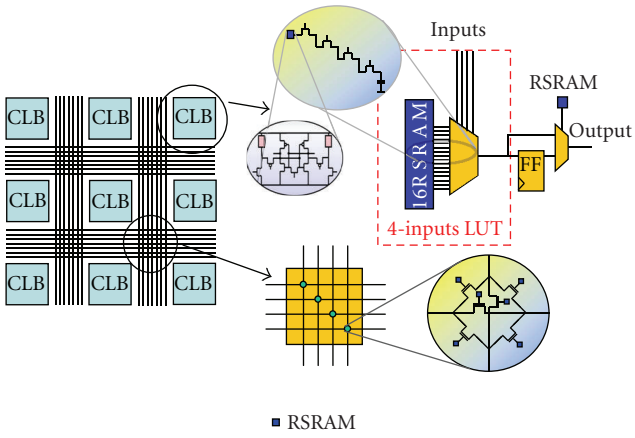


Figure 13: RSRAM-based elementary configurable logic blocks and switch matrix.

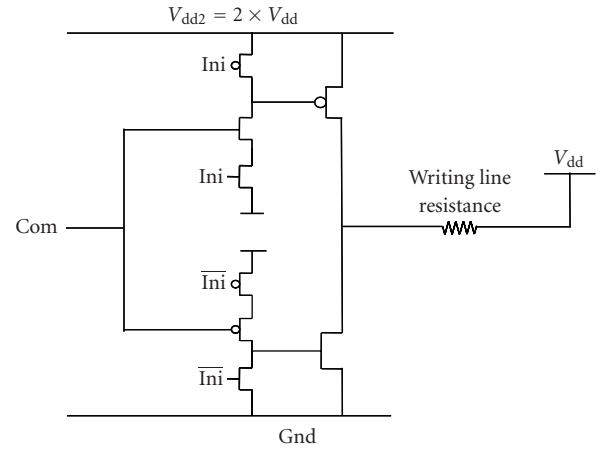


Figure 14: Bidirectional writing current generator.

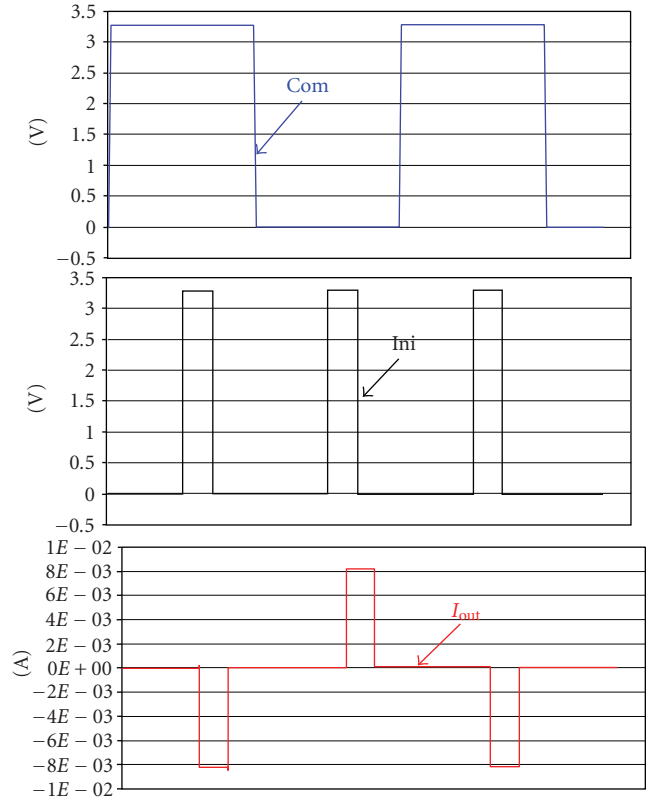


Figure 15: Output waveforms of the bidirectional writing current generator shown in Figure 14, obtained in 0.35 μm CMOS technology under a V_{dd} = 3.3 V.

The area overhead at the CMOS level is limited to one transistor per MRAM.

Loading information from the MRAM to the latch can be done very quickly, which means that the circuit can switch from one configuration to the other in a very short time. A and $\overline{A}$ signals are used to select one configuration or the other. It can be interesting, in future works, to combine these fast switching capabilities with the work of Kielbik [14]. Indeed, an FPGA with very fast reprogramming capabilities can be used to emulate a larger FPGA with a slower clock. This work proposes mechanisms to interface the different contexts since they are not necessarily independent from each other. In such circumstances, a context has to be loaded between each clock edge and flip-flop buffers have to be inserted to store the signals that have to be transmitted between the different contexts.

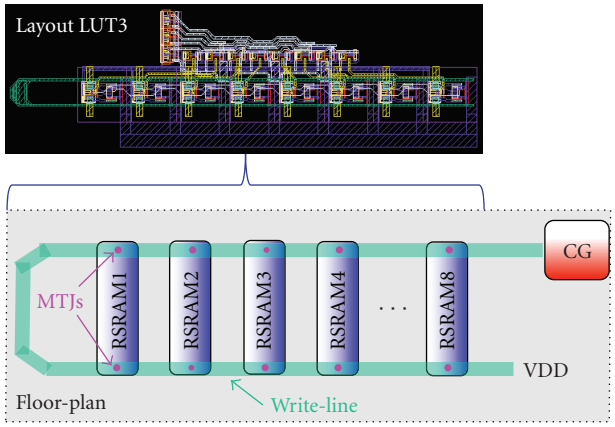


Figure 16: Floor-plan and layout of TAS-MRAM LUT3.

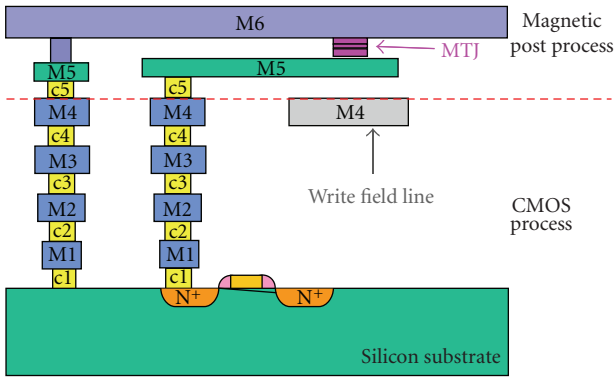


Figure 17: Cross-section of the stacking of the CMOS process and the magnetic postprocess.

4. Experimental Results of an LUT-4 Using Emulated FIMS-MTJs

A test chip has been realized to explore particular features of the RSRAM cell (Figure 10) and to validate the concept of the shadowed reconfiguration. On this chip (designed in a 0.35μm CMOS technology), RSRAM cells and RSRAM-based Lookup tables were implemented using transistors to emulate the switching resistances. Notice that this chip is just realized with CMOS technology and not with MRAM technology. For that sake, these transistors are biased in their linear region such that their resistance value can be controlled by the V_{gs} voltage. These transistors act then as resistance switching elements (RSEs).

As illustrated in Figure 8, the RSEs can be written without affecting the output of the RSRAM and then the output of the LUT, thereby providing a shadowed reconfiguration.

Figure 11 shows experimental results obtained from the RSRAM-based 4-inputs LUT, and where we can see the "Read" signal of the RSRAMs, the output of the 4-inputs LUT and the shadowed rewriting of the RSEs (two different profiles are stored: profile 1: 010101…, profile 2: 11110001…). Inputs of the LUT are swept thereby producing two different output voltage profiles depending on the values

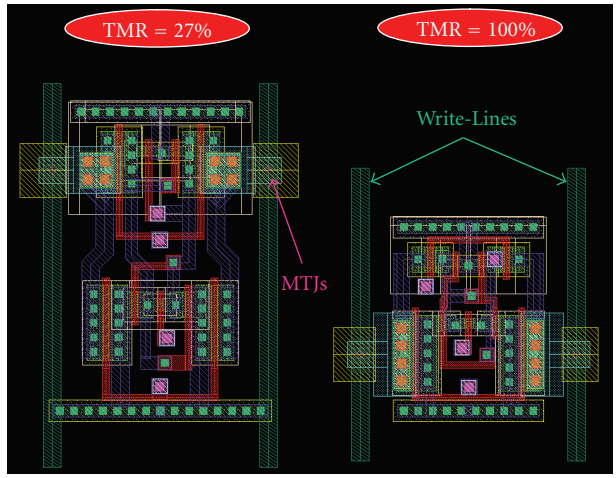


Figure 18: Implemented layout of the second TAS-MRAM cells (shown in Figure 6) validated for a TMR of 27 and 100%.

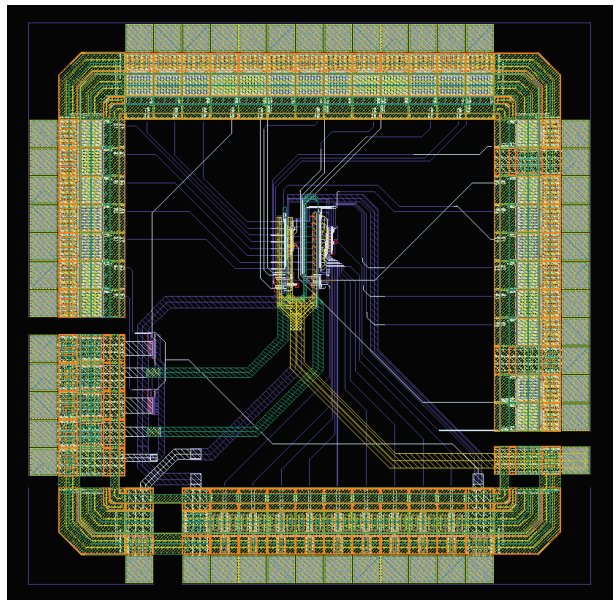


Figure 19: Implemented chip in 0.35μm CMOS technology combined with the MTJ-TAS magnetic postprocess.

stored in the RSEs. As can be noticed in Figure 8, the output of the LUT remains unchanged until a pulse is applied on the gate of the sensing transistor.

5. Simulation Results of the TAS-MRAM Cell

Simulations of the TAS-MRAM cells shown in Figures 5 and 6 have been carried out in 0.35μm technology combined with the model of the magnetic tunneling junction using the TAS writing scheme. A 1.4/0.7 and 2/0.7 (sizes are given in μm) have been, respectively, used for the W/L ratio for the N-ch (MN1 and MN2) and P-ch MOSFET transistors (MP1 and MP2) in the flip-flop except for transistor MN3 which must be sized properly in order

to achieve a compromise between the speeding up of the metastable transition and the correct functionality of the flip-flop. Thus a 0.7/0.7 has been used for the W/L ratio of transistor MN3. Regarding the TAS-MTJ parameters, a circular MTJ with 350 nm diameter, an RA product (i.e., resistance∗area) of 30 $\Omega\cdot\mu m^2$ has been considered and TMR ratio of 150%, which leads to resistance values of 312 Ω for R_{min} and 781 Ω for R_{max}. Figure 12 shows the output waveforms that are obtained from the first structure of the TAS-MRAM cell. The transistor sizing of the second structure has also been performed such that to achieve a good and stable functionality. The stability of the two structures versus a possible mismatch in transistors parameters has been evaluated through Monte Carlo simulations.

6. TAS-MRAM Based LUTs

In programmable devices, configuration bit stream is often stored in SRAM cells in order to configure digital blocks like LUTs, configurable logic blocks (CLBs), and interconnections between these blocks. We used this principle as a basis for TAS-MRAM-based Lookup table LUTs implementation. The structure of a TAS-MRAM-based CLB is shown in Figure 13, and in which an LUT-N is the main building block.

Lookup table's purpose is to implement Boolean functions. Truth table is stored in the TAS-MRAMs, and a multiplexing tree controlled by the inputs drives the data stored in the selected memory point into the output.

As shown in Figure 13, the structure of our TAS-MRAM-based LUTs is composed of a 2^NMRAM cells and a multiplexer circuit. We can also use TAS-MRAMs to drive the gates of pass transistors in the switch matrix and thus configure routing between inputs/outputs of the CLB blocks. Implementation of TAS-MRAM-based switch matrix and CLBs will be addressed within a future work.

A TAS-MRAM-based LUT-3 has been simulated in 0.35 μm CMOS technology in combination with a TAS writing scheme for the nonvolatile device, and has shown a good and stable functionality. The stability of the circuit has been evaluated, through Monte Carlo simulations, with regard to a possible mismatch on transistor parameters, that is, threshold voltage, sizing (W/L) that might degrade the TAS-MRAM operation, given the considered TMR value.

7. TAS-MRAM-Based Micro-FPGA Architectures

During the write operation, after the junction heating step, a current of a few mA is then applied on the write field line in order to write data in the magnetic tunneling junctions. The writing of "0" or "1" logic state depends on the applied current direction in the write line. This requires a bidirectional writing current. We propose hereafter a bidirectional current generator that can generate 8 mA in 0.35 μm CMOS technology during few nanoseconds. This current is quite important due to the distance between write field line and the MTJ. By reducing this distance, current will be reduced drastically (highly dependent of the technology process available). Its structure is shown in Figure 14. It consists in a three states current generator depending on the state of the control signals "Ini" and "Com." The output of this circuit is connected to the writing line, which is supplied with a voltage of V_{dd}. Hence in order to ensure a correct operation of the current generator, the latter must be used with a supply voltage of $2V_{dd}$. It operates as follows. When signal "Ini" equals 0 V, both transistors Q7 and Q8 are switched off. Consequently, there is no current flowing through the writing line. When "Ini" equals $2V_{dd}$, a current will then flow through the writing line in a direction that depends upon the state of the signal "Com." If "Com" equals 0 V, then the generated current in the writing line goes through the transistor Q7. On the other hand, when "Com" equals $2V_{dd}$, the writing current goes through the transistor Q8. Figure 15 shows the output waveforms of the proposed current generator. Therefore, we can see the control signals "Ini" and "Com" and the output current on the write line (I_{out}).

Figure 16 illustrates the floor-plan and the layout of MRAM cells in a real-time reconfigurable LUT-3 using such a current generator and the TAS writing scheme.

The writing line is implemented in a "U" shape such as allowing writing the data and its complement in the two MTJs of each MRAM cell. As described above, it is connected to the bidirectional current generator (depicted by "CG" in Figure 16). The configuration memory is addressable by both the selection heating transistor and the current flowing through the writing line, thereby allowing reconfiguration of the selected MTJs.

Figure 17 illustrates the cross-section of the stacking of the CMOS process and the magnetic postprocess. Figure 18 shows the implemented layout of the second TAS-MRAM cell validated for a TMR of 27 and 100%, the cells area being, respectively, about 160 μm^2 and 98 μm^2; three times bigger than SRAM cell. The TAS-MRAM cell (Figure 6) requires large transistors (MN4 and MN5) due to the high-heat currents (~2 mA) and, for this unbalanced structure, larger transistors MP3 and MP4 are needed (in order that their equivalent resistances Ron be smaller than the magneto-resistance ~312 Ω). However, by using the TAS-MRAM technology process improvement, MTJs' sizes will drastically decrease which allow not only lower heat currents (<0.3 mA) but also higher magneto-resistance (>3 kΩ) that induce smaller transistors sizes. Thanks to this improvement, the TAS-MRAM cell area could be equivalent in the future to the SRAM cell one.

The implemented TAS-MRAM-based micro-FPGA that combines 0.35 μm CMOS process and the TAS-MTJ magnetic technology is shown in Figure 19.

8. Conclusion

In this paper, we have presented the use of field-induced magnetic RAM (FIMS-MRAM) cells and MRAM cells with a thermally assisted writing scheme (TAS-MRAM) in nonvolatile FPGA circuits. The main advantage of the use of such a memory cell in FPGA circuits instead of the SRAM cell is its nonvolatile property thanks to the magnetic

tunneling junctions. This feature allows to power down the FPGA circuit in the standby mode when not in use and thus benefits to power consumption reduction. Moreover, it also benefits to configuration time reduction since there is no need to load the configuration data from an external nonvolatile memory as is usual in SRAM-based FPGAs. It has also been shown through experimental results of the RSRAM-based 4-inputs LUT using transistors to emulate the FIMS-MTJs that the use of such RSRAM cells allows to achieve shadowed reconfiguration during the FPGA circuit operation. Indeed, the resistance switching elements can be written, while the FPGA circuit is still operating thereby allowing a shadowed reconfiguration. The use of the TAS writing scheme instead of the FIMS one allows reduction of the required write current and thus the consumed power during write operation. Furthermore, in comparison to the FIMS writing scheme, it improves write selectivity and cell die area. Simulation results of the TAS-MRAM cell and a real-time reconfigurable LUT-3 based on this memory cell, in 0.35 μm technology combined with a TAS writing scheme, have shown a good and stable functionality.

Further works will be extended to 90 nm CMOS and 120 nm MRAM technology. Simulations have already been realized showing the structures robustness, but should be confirmed on silicon.

Acknowledgment

This project is funded by the National Agency of Research under Grant ANR-06-NANO-066 (CILOMAG).

References

[1] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992.

[2] http://www.actel.com.

[3] M. She, *Semiconductor flash memory scaling*, Ph.D. dissertation, University of California, Berkeley, Calif, USA, 2003.

[4] K. J. Hass, G. W. Donohoe, Y.-K. Hongt, B.-C. Choi, K. Degregorio, and R. Hayhurst, "Magnetic shadow RAM," in *Proceedings of the 7th Annual Non-Volatile Memory Technology Symposium (NVMTS '06)*, pp. 45–48, San Mateo, Calif, USA, November 2006.

[5] W. J. Gallagher and S. S. P. Parkin, "Development of the magnetic tunnel junction MRAM at IBM: from first junctions to a 16-Mb MRAM demonstrator chip," *IBM Journal of Research and Development*, vol. 50, no. 1, pp. 5–23, 2006.

[6] Freescale, June 2006, http://www.freescale.com.

[7] N. Bruchon, *Evaluation, validation and design of hybrid CMOS—Non Volatile emerging technology cells for dynamically reconfigurable fine grain architecture*, Ph.D. thesis, Université Montpellier 2, Montpellier, France, December 2007.

[8] I. L. Prejbeanu, M. Kerekes, R. C. Sousa, et al., "Thermally assisted MRAM," *Journal of Physics: Condensed Matter*, vol. 19, no. 16, Article ID 165218, 23 pages, 2007.

[9] A. Manchon, *Magnetoresistance and spin-transfer torque in magnetic tunnel junctions*, Ph.D. thesis, Université Joseph Fourier, Grenoble I, Grenoble, France, December 2007.

[10] W. Zhao, E. Belhaire, Q. Mistral, E. Nicolle, T. Devolder, and C. Chapper, "Integration of Spin-RAM technology in FPGA circuits," in *Proceedings of the 8th International Conference on Solid-State and Integrated Circuit Technology (ICSICT '06)*, pp. 799–802, Shanghai, China, October 2006.

[11] W. Zhao, E. Belhaire, B. Dieny, G. Prenat, and C. Chappert, "TAS-MRAM based non-volatile FPGA logic circuit," in *Proceedings of International Conference on Field Programmable Technology (ICFPT '07)*, pp. 153–160, Kokurakita, Japan, December 2007.

[12] W. C. Black Jr. and B. Das, "Programmable logic using giant-magnetoresistance and spin-dependent tunneling devices," *Journal of Applied Physics*, vol. 87, no. 9, pp. 6674–6679, 2000.

[13] N. Bruchon, L. Torres, G. Sassatelli, and G. Cambon, "Technological hybridization for efficient runtime reconfigurable FPGAs," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI: Emerging VLSI Technologies and Architectures (ISVLSI '07)*, pp. 29–34, Porto Alegre, Brazil, May 2007.

[14] R. Kielbik, *Efficient methods of ressource management in re-programmable systems*, Ph.D. thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 2006.