Targeting Tiled Architectures in Design Exploration

Lilian Bossuet[1], Wayne Burleson[2], Guy Gogniat[1], Vikas Anand[2], Andrew Laffely[2], Jean-Luc Philippe[1]

[1] LESTER Lab
University of South Brittany,
Lorient, France
{lilian.bossuet, guy.gogniat,
jean-luc.philippe}@univ-ubs.fr

[2] Department of Electrical and Computer Engineering
University of Massachusetts,
Amherst, USA
{burleson, vanand, alaffely}@ecs.umass.edu

Abstract

Tiled architectures can provide a model for early estimation of global interconnect costs. A design exploration tool for reconfigurable architectures is currently under development at LESTER-UBS. The tool allows various reconfigurable architectures to be compared for different applications and sets of constraints. One of the challenges of the tool is the ability to estimate interconnect costs at a high level of abstraction.

This project explores the use of the Adaptive System on a Chip (aSoC) tiled architecture, developed at UMASS as a target architecture for design exploration. aSoC provides an important capability to the LESTER tool by allowing interconnect costs to be modeled very early in the design process by partitioning and placing each portion of the computation into a square tile on a 2D grid. aSoC is primarily an interconnect architecture, based on static scheduling of virtual interconnects onto a highly characterized and regular physical interconnect fabric. aSoC supports a wide variety of cores, including dedicated, hardware programmable and software programmable thus allowing a wide range of design exploration.

1. Introduction

Design methodologies and system on chip (SoC) technologies will have to overcome several difficult challenges in the near future including the growing importance of on-chip interconnect. It is mandatory to improve the versatility and hence lifetime of a system to a greater extent in order to face the increasing costs of an advanced foundry process. Optimization of power consumption without affecting the necessary increase of computational performance is also a major issue for embedded systems. Design cycle reduction and very early performance evaluation especially of new architectures and heterogeneous architectures are also critical.

In order to face these challenges, it is necessary to provide novel approaches to design with higher quality and more rapidly at the system level, and to target new technologies. Thus, reconfigurable computing is becoming very attractive in SoC design since it provides an efficient executive platform that overcomes several technological issues. Moreover, heterogeneous domain specific reconfigurable architectures may be more appropriate for multimedia and telecommunication applications than classical RISC or DSP processors or FPGAs since the parallelism and granularity of resources do not lead to an efficient performance/power trade-off. Performance improvement and power reduction can benefit from the wide spatial parallelism typical of reconfigurable architectures in addition to temporal parallelism (i.e. pipelining). The lifetime of a system can be enhanced through reconfiguration and versatile environment evolutions can be supported through dynamic reconfiguration mechanisms (Run Time Reconfiguration). At the circuit level, improvement of the reliability is also expected - thanks to a very regular technology, which is important to support fault avoidance in deep sub-micron technologies. Finally, regular SoC platforms allow the use of aggressive circuit techniques for interconnect that would be too risky to use widely in a conventional ASIC methodology.

To efficiently utilize reconfigurable computing it is necessary to propose new methodologies that emphasize the synergy between the application and the architecture (e.g. granularity, parallelism). In order to manage the wide design space that provides reconfigurable architectures, it is important to define rigorous exploration strategies both at the system and at the architectural levels to progressively converge toward an efficient solution. These methodologies should be based on several models to facilitate rapid evaluation and support architecture and technology migrations.

Hence, at the system level (i.e. algorithmic level) the representation should exhibit the intrinsic application optimization potential. At the architectural level, the

representation should describe not only the computation and memory resources but also the communication resources. This last feature is very important in the case of reconfigurable architectures since interconnect costs are very difficult to estimate early in the design cycle, which can ultimately lead to a violation of application constraints. In our approach, we overcome the complexity of estimating routing through an efficient communication topology, which is tile-based since this architecture model has proven its efficiency in terms of locality and regularity. Moreover, tile-based architectures present the strong advantage of deterministic communication topology and performance.

Run time reconfiguration has to be evaluated early in the design cycle in order to define a scheduling of context switching and to build an architecture that enhances the flexibility. In our approach the considered target is the aSoC architecture that supports static communication scheduling in order to adapt the communication resources to the execution schema. Moreover the proposed exploration methodology is able to define multi context tiles. In this paper we propose an original exploration flow that targets aSoC architecture. The definition of the architecture and the associated tools has been done considering the challenges presented above and in our work, we take a longer term approach leveraging trends in VLSI which will allow much larger systems to be integrated on a chip.

This paper is organized as follows. Section 2 describes several works dealing with tile-based architectures and design space exploration methodologies. Section 3 presents the tile-based architecture - aSoC. Section 4 describes the design space exploration flow. Finally, the section 5 concludes this paper and exposes several future directions.

2. Related work

Reconfigurable architectures are becoming very popular and many researchers are focussing on this topic [1]. All architectures can be characterized by their interconnect resources but in the case of reconfigurable architectures this feature is particularly important since interconnections represent a large ratio in time performances. As FPGAs, interconnect resources can also become a bottleneck for reconfigurable architectures.

Many works on reconfigurable architectures aim to define efficient interconnect styles. Among these are some target tile-based architectures that are characterized by a distributed interconnection overlaid on a set of heterogeneous, or non-heterogeneous resources. Furthermore, the interconnect structure is independent of the connected resources. The adaptive System On a Chip (aSoC) architecture [2], the Raw Microprocessors architecture [3] or the Field Programmable Function Arrays (FPFA) architecture [4] are good examples of tile-based architectures. These architectures are energy-efficient, in particular for digital signal processing [4, 5, 6] applications.

In this paper, the aSoC tiled architecture, developed at the University of Massachusetts Amherst [2], has been presented as the target architecture for design space exploration. The problem of design space exploration consists of the proper selection of design alternatives in order to meet a given design goal [7]. Several works have focused on this issue, with several levels of granularity. In [8] FPGAs are targeted and the exploration is based on precise estimations. For the cluster-based architecture DART, a dynamically reconfigurable architecture [9] proposes a design flow able to take into account dynamic reconfiguration. In this design flow, the application is specified using C at a high level of abstraction, in order to exhibit application's parallelism. Their flow is based on the SUIF compiler from Stanford University to generate a control and data flow graph (CDFG).

In [7, 10] the design space exploration targets a mesh architecture called KressArray - a fast reconfigurable ALU. The exploration tool, Xplorer aims to assist the designer in finding a suitable architecture for a given set of applications. This tool is architecture-dependent, but the use of fuzzy logic to analyze the results of the exploration is a very attractive approach.

[11] presents the design space exploration for the Raw Microprocessor as an example for a tiled architecture. The Raw Microprocessor is reminiscent of coarse-grained FPGA and comprises a replicated set of tiles coupled together by a set of compiler orchestrated, pipelined, switches. Each tile contains a RISC-like processing core and SRAM memory for instructions and data.

First challenge, as pointed out in [11], in this architecture is to determine the best division of VLSI resources among computing, memory, and communication - the *balance problem*. The second challenge is to determine the number and granularity of the tiles – the *grain problem*. To overcome these challenges, the authors propose an analytical framework, which is based on several models: application model, architecture model and cost model. For the cost model, the processor and the communications are taken into account. The framework in this case however is tile-dependent as all tiles are identical - RISC cores and SRAM memory.

In this paper, we proposed a design space exploration method targeting the aSoC architecture. For this work we have considered the major characteristics noted previously. The proposed method uses a high level specification of the application in C language as the input. This is parsed into an intermediate representation - a hierarchical control and data flow graph (HCDFG).

For the architecture description, two models have been formalized, one to describe the tiles with several level of granularity and one to describe the global architecture

with communication resources. A distinguishing characteristic of the aSoC architecture is that it can have heterogeneous tiles. The proposed method is both tile and architecture-independent since it is based on two generic architecture models. Moreover dynamic reconfiguration can be taken into account during the exploration process. This can be carried out when targeting run-time reconfiguration.

3. aSoC architecture

aSoC is a modular communications architecture and serves as a platform for on-chip interconnects. Its tiled architecture addresses both scalability and flexibility. As shown in figure 1, aSoC contains a two dimensional mesh of computational tiles each of which consists of a core and an associated communications interface (figure 2). The interface design can be customized based on core data widths and operating frequencies to allow for efficient use of resources. Communication between nodes takes place via pipelined, point-to-point connections. An important feature of aSoC architecture is its ability to support both statically scheduled and run-time dynamic transfer of data [2, 6].

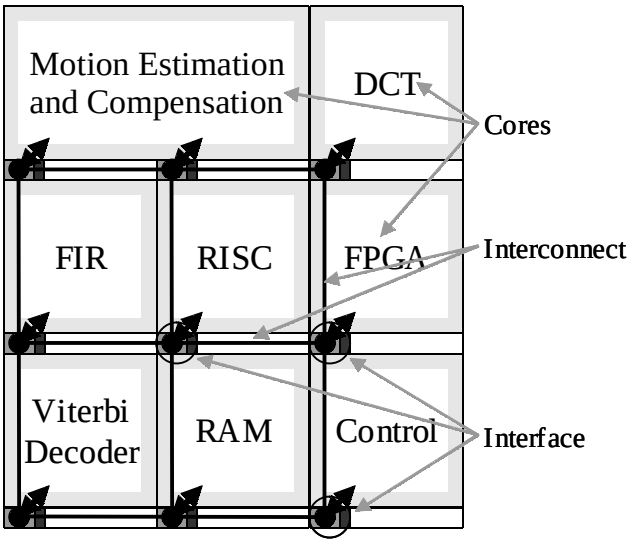

Figure 1. aSoC Architecture

A high level view of the Communication Interface has five components responsible for aSoC communications [12]. These are: *Interface Crossbar* – allows for inter-tile and tile to core transfer; *Interconnect/Instruction Memory* – contains instructions to configure the interface crossbar on a cycle-by-cycle basis; *Interface Controller* – control circuitry to select an instruction from the instruction memory; *Coreport* – data interface and storage for transfers to/from the tile IP core; *Communication Data Memory* – buffer storage for inter-tile data transfer.

The architectural framework has been designed and simulated in both 250nm and 180nm technologies. Tiles are square, and can be up to 30k times the minimum feature size, while maintaining interconnect clock speeds of up to 400MHz, in the 180nm system. The communications interface and interconnect mesh infrastructure use approximately 20K transistors per tile, and occupy less then 10% of chip active area. Each core is constrained to fit in within one or more tiles in a row of the aSoC.

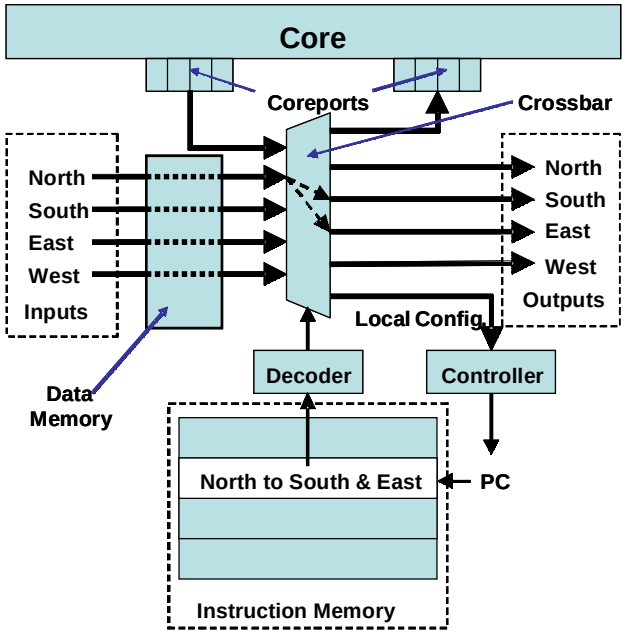

Figure 2. Detailed Communications Interface

4. Design space exploration flow

4.1 Proposition of a Design Flow

Reconfigurable tile-based architectures have many characteristics that must be considered in order to perform an efficient design space exploration. Tile interconnect styles, resource (computing or memory) granularity and corresponding numbers, architecture topology, tile types and numbers of each tile are for example different characteristics that must be settled. These architecture characteristics define a very large design space that can rapidly become complex to explore manually in order to converge to an architecture and meeting a set of application constraints. Hence, the architect may face the following questions:

- Among modeled tiles, what are the best tiles for the application with a set of constraints?
- In order to realize the application, how many tiles are necessary?
- What is the optimal tiles placement in the global architecture?

- What interconnect style improves the application's performance?
- How to schedule the application communications?

In order to answer these questions, we have developed a design exploration flow targeting tiled architecture (figure 3). This is composed of several steps, which are as under:

First, the system level specification is provided in a high level language (C language) and is then translated into an intermediate representation - the HCDFG model. This model is a Hierarchical Control and Data Flow Graph allowing efficient algorithm characterization and exploration of complex modern applications including control flow and multi-dimensional data. Further descriptions of the HCDFG model are presented in section 4.2.

After the **Application Analysis step** during which the application is characterized and scheduled, the **Tile Exploration Step**, aims at estimating and compare each function of the application on several tiles (e.g. ASIC, FPGA, DSP, RISC) in order to exhibit the best tile for one or several functions. Note that in the case of reconfigurable tiles when a tile is able to support several functions, each one is executed separately through a specific execution context that is managed at run time (Run Time Reconfiguration).

During the tile exploration step, all the performance estimation results are gathered into a table where each function of the application is characterized for at least one tile. The corresponding tiles can be designed for the current application or can already exist in a library. The selection of the best tiles for one application can be done manually after an analysis of the exploration/estimation results or can be selected automatically during the **aSoC Builder Step**. This step aims to build an aSoC architecture and map the application onto it. Hence, starting from the application specification and a library of characterized tiles, the aSoC builder step ends with a proposition of aSoC (allocation of tiles and placement of the selected tiles into the aSoC) and a mapping of the application on the architecture (partitioning and scheduling).

In order to completely build the aSoC architecture it is necessary to perform the communication synthesis step. In the aSoC case, it is carried out through a static communication scheduling step that computes the instructions that configure the interface crossbar (figure 2) on a cycle-by-cycle basis.

The last step - **Analysis Step** aims to make a complete estimation of the proposed solution in order to give the designer a quality measure of the final solution

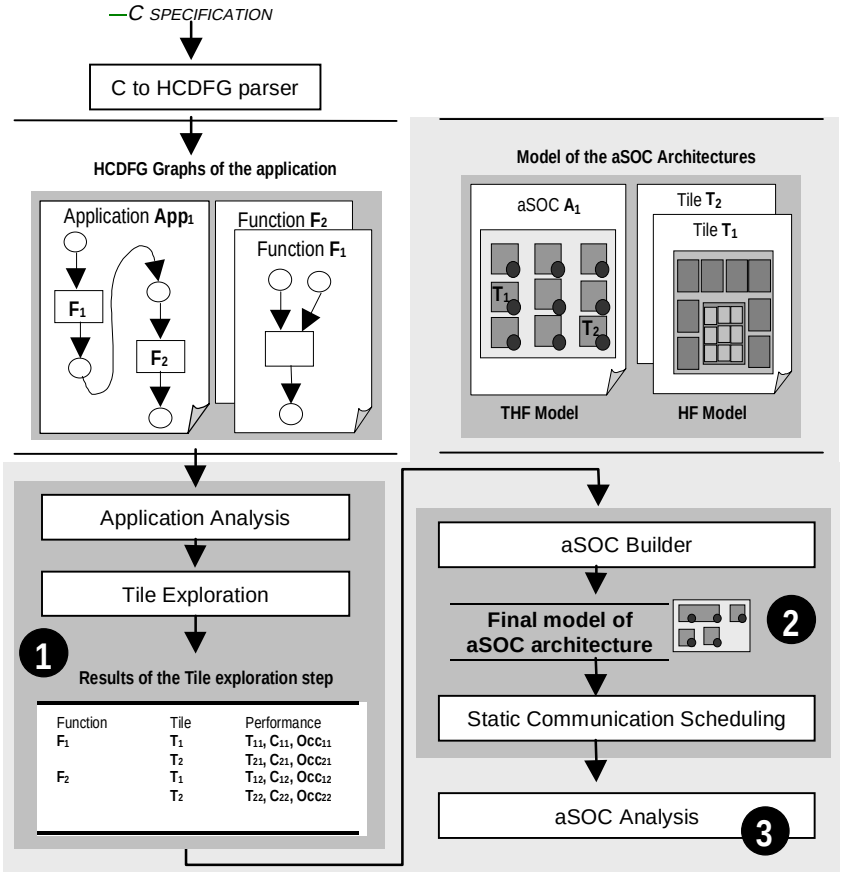

Figure 3. Design exploration flow targeting tiled architecture.

(adequacy, performance, flexibility). In this step, the results of the partitioning and the communication scheduling are used to estimate the global execution time and the global consumption. The quality of the aSoC architecture can also be characterized with information like utilization rates of each tile, size of the tiles, width of the connections, utilization of the connections etc.

To propose an efficient and flexible exploration flow two main features must be considered: first, the definition of the flow must be based on a high level application representation to take benefit of the optimization potential that is exhibited at the system level. Second the architecture must be defined through a formalized model in order to enhance the genericity of the description and be technology independent. Hence, we have considered a model centric approach, which is based on two modeling processes: a *reconfigurable architecture modeling* to model the tiles and a *tile based architecture modeling* to model the aSoC. Following paragraphs elaborate on each part of the design flow:

4.2 Application Specification and Analysis

The application to be designed is specified with the C language and can be composed of several C functions. Once the application has been functionally validated, it is translated into an intermediate format which is a Hierarchical Control and Data Flow Graph (HCDFG) [13]. In HCDFG, each C function in the specification corresponds to a node at the top level of the hierarchy, as shown on figure 4. Hence, a C function is a node encapsulating an HCDFG. An application is modeled with several nodes (which could be as many as the number of C functions) connected through control structures or dependence relations (typically represented with a CFG Control Flow Graph).

An HCDFG is therefore a graph, which can contain other HCDFGs (other functions), control structures (CFG) encapsulating HCDFGs or DFGs (Data Flow Graph) and DFGs. This decomposition enables to precisely highlight the hierarchy of the application, which is very important

```
void uppol2 (short AH1, short AH2,
short PH, short PH1, short PH2, short
*APH2) {

    short const_128  = 128;
    short const_m128 = -128;
    short const_35512    = 35512;
    short const_12288    = 12288;
    short const_m12288   = -12288;
    short tmp1,tmp2;
    short WD1,WD2,WD3,WD4,5;

1   tmp1 = AH1 + AH1;
    WD1 = tmp1 + tmp1;
2   if ( (PH>>15) == (PH1>>15) )
        WD2 = 0 - WD1;
    else
        WD2 = WD1;
3   if ( (PH>>15) == (PH2>>15) )
        WD3 = const_128;
    else
        WD3 = const_m128;
4   tmp2 = WD2>>7;
    WD4 = tmp2 + WD3;
    WD5 = AH2 * const_35512;
    *APH2 = WD4 + WD5;
}
```

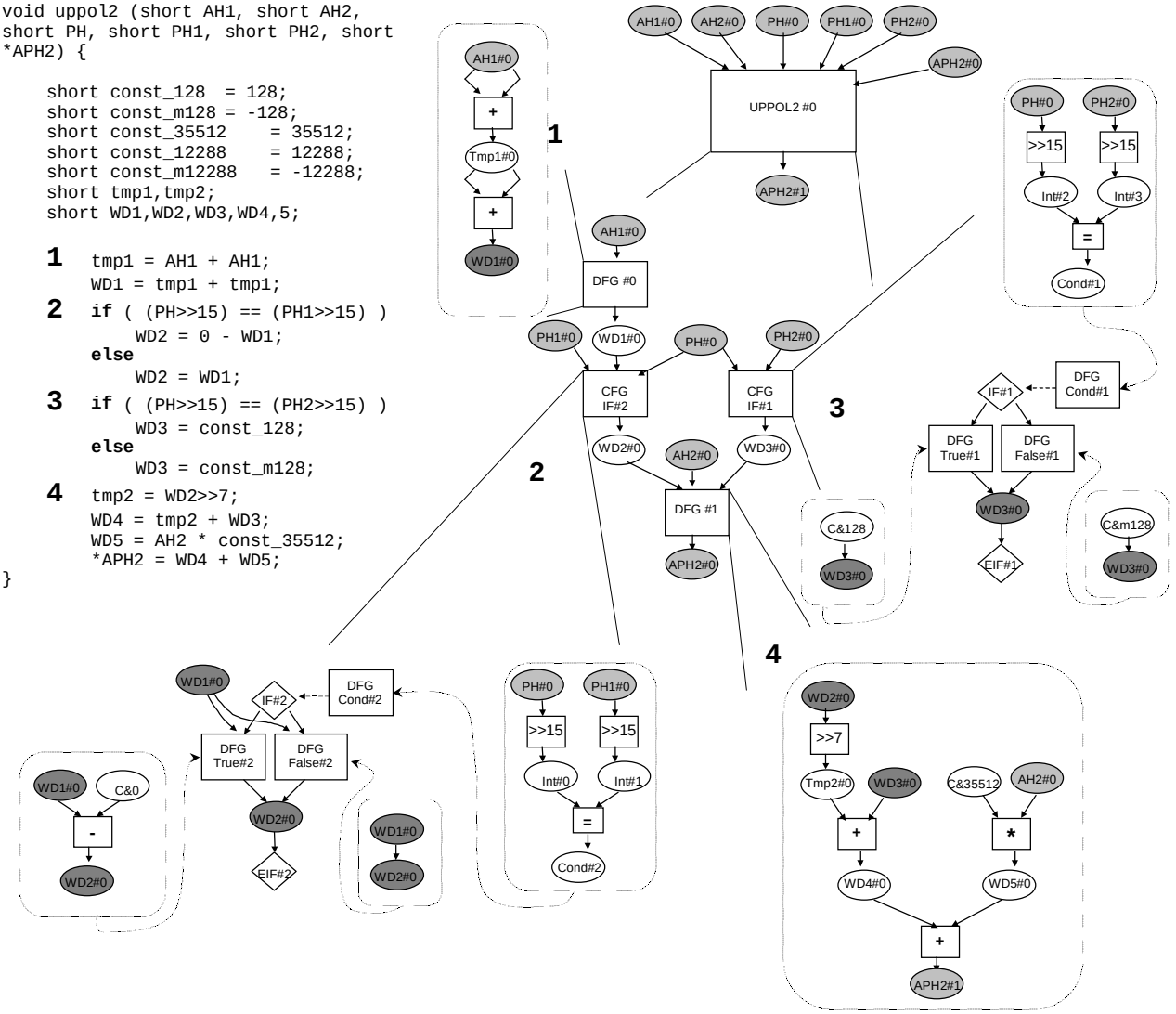

Figure 4. C to HCDFG model

for exploration. For example it enables to consider multi granularity during the exploration flow. A CFG only contains elementary control nodes (e.g. loops, conditional branching) and HCDFGs or DFGs. A DFG corresponds to linear sequences of operations. A DFG only contains elementary memory and processing nodes. Namely it represents a sequence of non-conditional instructions. There are two kinds of elementary nodes in a DFG whose granularity depends on the underlying architectural model: a processing node represents an arithmetic or logic operation (ALU, MAC, +, -, etc.). A memory node represents a data transfer. Its parameters are the transfer mode (read/write), the data format and the memory hierarchy level, which can be fixed by the designer (cache, in/off core memory). Memory nodes are explicitly represented in the HCDFG model in order to be able to efficiently deal with applications requiring data intensive manipulations, such as image processing.

During the **Application Analysis Step** (figure 3, number 1) each function (i.e. HCDFG) is considered independently. However, during the **aSoC Builder Step** (figure 3, number 2) the application (i.e. CFG), is considered as a whole in order to build the complete architecture. During the application analysis step, the application's orientation is defined through three axes - processing, control and memory. Specific metrics are used to find this orientation. These are described in [14]. Once the application orientation is found a resources scheduling is performed accordingly. For example, if the application is processing oriented, the first step is process scheduling and the second step is memory scheduling (the opposite if the application is memory oriented). The main objectives of the application analysis step are to highlight the intrinsic processing and memory parallelisms of the application and to give some guidance on how to build the architecture (pipeline, parallelism, memory hierarchy etc.).

4.3 Architecture Modeling

Two models are necessary for the exploration flow - one to model the tiles, and the other to model the aSoC architecture. In the first case it is necessary to describe a tile by what it is able to perform in terms of processing and memories. This is *functional modeling* [15]. In the second case it is necessary to describe the geographical placement of tiles and the characteristics of interconnect resources (width, number, style etc.). This is *physical modeling*. These two models are called the **HF** (Hierarchical Functional) **model** and the **THF** (Tile Hierarchical Functional) **model** [16].

With the **HF model** [15, 16] the architecture's elements are functionally described and the modeling allows representing different architectural styles and different architecture elements. However, the routing resources are not explicitly described, even if they are taken into account in the estimation tool. In fact, connection resources are taken into account by connection costs in the HF model. There are two types of elements in this model: the hierarchical elements and the functional elements. The hierarchical elements are used to describe the architecture's hierarchy. A hierarchical element can be composed of functional elements and other hierarchical elements. The functional elements are used to describe the architecture's resources. They can be logical resources, input/output resources, memory resources, computing resources etc.

The **THF model** [16] is an extension of the HF model since the HF model is insufficient to model tile-based architectures, where the communications between the tiles must be explicitly modeled. As opposed to the HF model, the THF model uses a physical description to model the tiles. This time, each tile is described by its position in a two dimensions space (X-axis, Y-axis). This way, it is possible with this model (for an automatic tool or manually) to perform the placement of the tiles. Note that, each tile still has its internal description specified with the HF model. The communications between the tiles are described by the possible inter-tile interconnections and by the type of communication supports (wire, bus, crossbar). Moreover the communication topology is characterized by a cost function, which depends on the coordinates of the tiles.

4.4 Tile Exploration Step

The tile exploration step (figure 3, number 1) uses a library of tiles modeled with the HF model, and the description of the functions (HCDFG formalism) as inputs to evaluate. To perform a tile exploration, the user has the opportunity to define new tiles based on the guidance obtained after the application analysis step or to use existing ones that he or she has already built. During that step the central tool - Relative Estimation Tool estimates the function performances for each tile selected from the library and proceeds in following three steps:

The **projection** step makes the correspondence between the necessary resources (processing and memory) of the function and the available resources of the tile. At the end some resources of the tile are assigned to the functions. If a difference of resource granularity exist between the necessary resources (from the application) and the available resources (in the tile), the necessary resources can be decomposed as fine-grained resources using a library called Technological Trees (the same method is used with the PipeRench reconfigurable architecture [17]). The method used to make the assignment between necessary and available resources is to consider the communications between all the processing and memory resources of the application, in order to assign in the same hierarchical level of a tile the

resources that communicate the most. In order to perform the mapping we use a heuristic based on the Weighted Bipartite-Matching Algorithm [18].

The **compositing** step takes into account the function's scheduling in order to refine previous estimations, since it is necessary to add resources dedicated to realize the scheduling like multiplexer, register or states machine. These additional resources are taken into account in the last step, the estimate process.

Finally, the **estimate** step characterizes the specified function performances implemented into the modeled tile. The estimations take into account the static costs of the tile model (interconnect costs between two hierarchical elements and the usage costs of the functional elements), and the dynamic costs of the function (critical path, operators communications, memories reads/writes). The results of this step are gathered into a table where each function of the application is characterized for at least one tile. It is also possible to have several tables for one function (and one tile) in the case of architectural exploration, then one table is produced for each function's scheduling. The **Application Analysis** step proposes several scheduling for a same function (so several parallelism levels), since for each potential time constraint a scheduling is defined. For example, if two time constraints are considered for a single function then two result tables are defined. This feature enables to explore the application's parallelism at a high level and to make a projection on a specific tile.

4.5 aSoC Builder

This step is carried out using the aSoC application-mapping environment, *AppMapper* [12], This environment builds upon existing compiler infrastructure provides an automated flow with the potential for user interaction. This flow roughly consists of three steps as shown in Figure 5. The first step uses SUIF [19] to build an optimized syntax tree representation of the code. During this stage, users are allowed to annotate branches of the syntax tree with desired target core types. Branches of this syntax tree form *Basic Blocks* of code to be placed in specific aSoC cores. The second step performs the analysis at the heart of the aSoC Builder concept in two distinct parts. First, the *Basic Blocks* are assigned to the best fit core type based on *Run Time Estimations*. After this assignment a greedy algorithm spreads the *Basic Blocks* over the available cores within the same type. The final core selection attempts to maximize system throughput and is subject to inter-block dependencies. For example, blocks of code assigned to the RISC processor core type would be placed in the available RISC processors. Placement can be accomplished at this stage as the aSoC architecture features predictable low cost core interconnections between cores. It is during this *Assignment* step that aSoC Builder can specify, modify or iterate on the distribution of core types in the system. The last step in the AppMapper process uses a combination of tools and compilers to generate the core specific binaries and the required communications instructions.

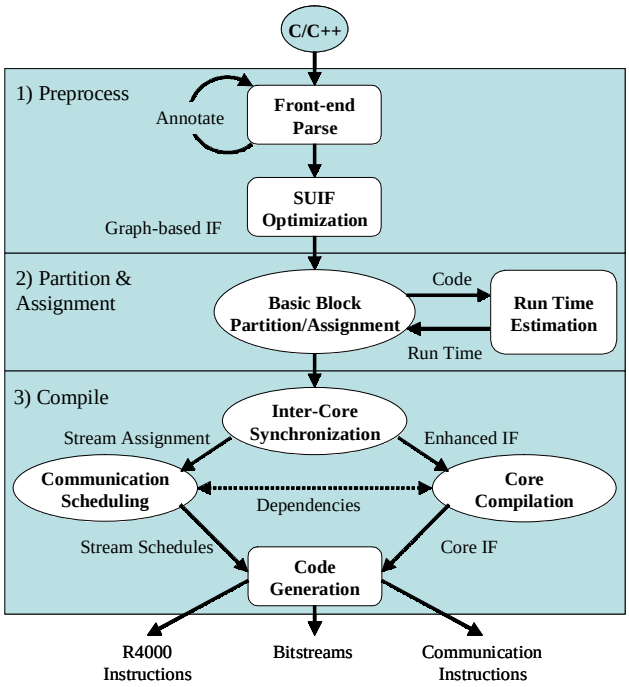

Figure 5. aSoC application mapping flow

4.6 Analysis Step

Analysis Step aims to make a complete estimation of the proposed solution. In this step, the results of partitioning and the communication scheduling are used in order to estimate the global execution time and the global power consumption. It is also possible to carry out an analysis of the results in order to evaluate the quality of the solution (utilization rates of each tile, size of the tiles, width of the connections, utilization of the connections etc.). The utilization rates of each tile depend of the tile; Utilization in FPGAs refers to occupancy of logic blocks, while utilization in an instruction set processor refers to the percentage of available cycles used. Note that clock and voltage scaling allow low utilization to be exploited with power savings.

5. Conclusions and future work

A prototype of the aSoC interconnect approach outperforms the standard IBM CoreConnect on-chip bus protocol by up to a factor of five in both single word and burst transfer modes [2]. The point-to-point nature of this architecture also allows a predictable and scalable communication clock speed of 400 MHz in 0.18 μm

technology [12]. Most recently some basic matrix operations such as matrix transpose, matrix-vector multiplication and sparse matrix-vector multiplication have been mapped and simulated on the aSoC in order to explore the behavior for very regular and scalable applications. The results of these experimental studies establish the ability of the aSoC to carry out fairly complex communications between the tiles and confirm its scalability and adaptability.

In order to exploit efficiently the potential of the aSoC architecture we have defined an original design exploration flow working at a high level of abstraction. It enables to compare several reconfigurables architectures in order to find the best target for one or several applications. The exploration can be carried out at the tile or at the aSoC level that conducts to a large architectural exploration.

Future work includes the exploration of larger and more heterogeneous systems. Our largest benchmark currently under study is a large satellite communication System-on-Chip currently implemented with 6 distinct Tensilica RISC cores and various dedicated blocks totaling 200 million transistors. This work will further provide insight into the ability of aSoC to carry out DSP and control for voice, video and data processing over large networks. We expect to develop a larger set of Design Exploration benchmarks and integration with other tools for specific optimizations such as memory, multi-processor compilation and power-aware computing. Finally, techniques for test and fault-tolerance and their associated overhead will also be an objective of design exploration.

We close with the conjecture that targeting tile-architectures may provide a reasonable early estimate of global interconnects even when the final target architecture is not tiled. This is similar to using FPGAs to estimate costs for an ASIC implementation.

6. Reference

[1] R. Hartenstein. A Decade of Reconfigurable Computing : a Visionary Retrospective. *In IEEE Design , Automation and Test in Europe Conference, DATE'01, Munich, Germany, 13-16 March, 2001.*

[2] J. Liang, S. Swaminathan, R. Tessier. aSoC : A scalable, Single-Chip Communications Architecture. *IEEE Conference on Parallel Architectures and Compilation Technique, Philadelphia, USA, October 2000.*

[3] E. Waingold and all. Baring it all to Software. *IEEE Computer, September 1997, pp. 86-93.*

[4] P. M. Heysters, J. Smit, G. Smit, P. Havinga. Mapping of DSP Algorithms on Field Programmable Function Arrays. *In IEEE Field Programmable Logic, FPL'00, Australia, August 2000.*

[5] J. M. Rabaey. Reconfigurable Processing: The Solution to Low-Power Programmable DSP. *In IEEE International Conference on Acoustics Speech and Signal Processing, ICASSP'97, Munich, Germany, April 1997.*

[6] A. Laffely, J. Liang, P. Jain, N. Weng, W. Burleson, R. Tessier. Adaptive System on a Chip (aSoC) for Low-Power Signal Processing. *Asilomar Conference on Signals, Systems and Computers, November 2001.*

[7] U. Nageldinger. Coarse-Grained Reconfigurable Architecture Design Space exploration. *Ph.D. Thesis, University of Kaiserlautern, Germany, June 2001.*

[8] S. Bilavarn, G. Gogniat, J.L. Philippe. An Estimation and Exploration Methodology from System-Level Specifications: Application to FPGAs. *In ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA'03, Monterey, CA, USA, February 2003.*

[9] R. David, D. Chillet, S. Pillement, O. Sentieys. DART: A Dynamically Reconfigurable Architecture dealing with Future Mobile Telecommunications Constraints. *In IEEE Reconfigurable Architectures Workshop, RAW'02, Fort Lauderdale, USA, April 15, 2002.*

[10] R. Hartenstein, T.H. Hoffmann, U. Nageldinger. Design-Space Exploration of Low Power Coarse Grained reconfigurable Datapath Array architectures. *In International Workshop on Power and Timing Modeling Optimization and Simulation, PATMOS 2000, Göttingen, Germany, September 13-15, 2000.*

[11] C. A. Moritz, D. Yeung, A. Agarwal. Exploring Optimal Cost-Performance designs for Raw Microprocessors, *In IEEE Symposium on Field Programmable Custom Computing Machines, FCCM'98, Napa, CA, USA, April 1998.*

[12] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier. An Architecture for scalable On-Chip Communication. *Technical Report, University of Massachusetts, Amherst, September 2002.*

[13] J. P. Diguet, G. Gogniat, P. Danielo , M. Auguin, J.L Philippe. The SPF model. *In Forum on Design Language, FDL'00, Tübingen, Germany, September, 2000.*

[14] Y. Le Moullec, N.Ben Amor, J.P. Diguet, M. Abid, J.L. Philippe. Multi-Granularity Metrics for the Era of Strongly Personalized SOCs. *In IEEE Design, Automation and Test in Europe, DATE 03, Munich, Germany, 3-7 March, 2003.*

[15] L. Bossuet, G. Gogniat, J.P. Diguet, J.L. Philippe. A Modeling Method for Reconfigurable Architecture. *In IEEE International Workshop on System On a Chip, IWSOC'02, Banff, Canada, July, 2002.*

[16] L. Bossuet, G. Gogniat. Reconfigurable Architecture Modeling: The THF Model, Application to the aSoC Architecture. *Technical Report, University of South Brittany, September 2002.*

[17] M. Budiu, S. C. Goldstein. Fast Compilation for PipeRench Reconfigurable fabrics. *In ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA'99, Monterey, CA, USA, 1999.*

[18] D. Gajski, N. Dutt, A. Wu, S. Lin. *High-Level Synthesis. Introduction to Chip and System Design.* Kluwer Academic Publishers, 1992.

[19] R. Wilson, R. French J. Hennessy. *SUIF: An Infrastructure for Research on Paralleling and Optimizing Compilers*, In ACM SIGPLAN Notices, December 1996.