www.czasopisma.pan.pl PAN www.journals.pan.pl
INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2011, VOL. 57, NO. 2, PP. 177–183
Manuscript received October 18, 2010; revised April 2011. DOI: 10.2478/v10177-011-0025-8

Methodology for Implementing Scalable Run-Time Reconfigurable Devices

Łukasz Kotynia, Piotr Amrozik, and Andrzej Napieralski

***Abstract*—The aim of this paper is to present the implementation methodology for an ASIC constituting the fine-grained array of dynamically reconfigurable processing elements. This methodology was developed during the work on a device which can operate as a typical Field Programmable Gate Array (FPGA) with some bio-inspired features or as a multi-core Single Instruction Multiple Data (SIMD) processor. Such high diversity of possible operating modes makes the design implementation extremely demanding. As a consequence, the comprehensive study and analysis of the different possible implementation techniques in this case allowed us to formulate a consistent and complete methodology that can be applied to other systems of similar structure.**

***Keywords*—design reuse, floorplanning, FPGA, implementation, verification, reconfigurable logic, timing analysis.**

I. Introduction

IT seems fairly natural that realisation of any IC design is determined by its structure. Consequently, each new approach to a design architecture results in a need of an appropriate implementation path. The goal of this paper is to present a reliable implementation flow that addresses all the characteristics of the specific organization of a reconfigurable design to produce a ready-to-manufacture product. Unlike commercial methodologies, this paper presents an open and accessible way of developing IC prototypes for programmable devices. We take the semi-custom approach based on standard logic cells. One can include some full-custom elements to the data path to gain speed and/or reduce silicon area. However, it is not in the scope of this paper. Our objective was to create a firmly structured implementation path that can be easily rerun. It was an important aspect taking into account a high number of essential changes in the behavioural description of the chip done during the work on the project. In our approach we refer to a complete implementation path rather than to separated front- and back-end stages. This way we concentrate on getting a global solution for the entire process instead of intermediate steps. The presented methodology is based on combining Electronic Design Automation (EDA) tools seamlessly together using standard file formats and TCL based tools mostly from Cadence Design Systems but also other vendors.

The novelty of the proposed flow lies in a combination of different techniques rather than developing new software tools, models, etc. Moreover, we are employing automated functional verification as a part of the final sign-off stage. The combination of constrained random testbench generation with annotated delays brings an effective way to overcome limitations of using Static Timing Analysis (STA).

The paper is organised as follows. The second section briefly presents the architecture of the PERPLEXUS chip with special emphasis on issues that we found most important in terms of implementation. The section III constitutes a theoretical background. It is focused on the issues directly related with the implementation. This part of the paper is intended to give a justification for further design and implementation decisions. The section IV presents the proposed implementation flow divided into three main steps: getting the netlist, getting the final layout, and timing validation. The proposed methodology validation is discussed in the section V. Finally, the paper is concluded with some general remarks in the final section.

This work was supported by the European Union (PERPLEXUS project, Contract No. 34632).

All authors are with the Department of Microelectronics and Computer Science of Technical University of Lodz, Poland (e-mails: lkotynia@dmcs.p.lodz.pl; pamrozik@dmcs.p.lodz.pl; napier@dmcs.p.lodz.pl).

II. Architecture Overview

The methodology described in this paper was elaborated within the framework of the PERPLEXUS project [1]. The main aim of this project was to develop a scalable hardware platform made of custom reconfigurable and, as stated in the project, ubiquitous computing modules with some bio-inspired capabilities. The run-time reconfigurable (RTR) ASIC, called Ubichip was the main component of these modules. Its architecture allows implementing specific bio-inspired mechanisms like dynamic routing [2] or self-replication [3]. Additionally, it supports emulation of Spiking Neural Networks (SNN) with Address Event Representation (AER) scheme [4]. The circuit merges functionality of a novel FPGA device and a multi-core processor. In this paper we concentrate mainly on hardware realisation of the Ubichip. Therefore, we are not covering mentioned mechanism in detail.

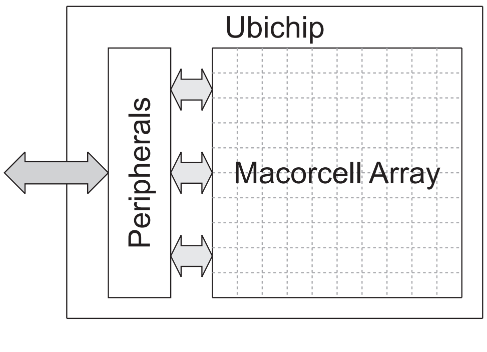

Fig. 1. System architecture of the Ubichip.

From the system point of view, the mentioned ASIC comprises an array 10 by 10 of configurable elements called Macrocells and some peripherals including modules responsible for configuration, control, and communication with the external world – see Fig.1. In this paper we focus on the array of the Macrocells, since it is the key part of the chip. The array of the Macrocells can operate in two different modes. In the first mode it acts as an FPGA device designed to facilitate bio-inspired behaviours. In the other mode, each Macrocell can be seen as a 16-bit processor which makes the array a multi-core processor operating in a Single Instruction Multiple Data (SIMD) manner.

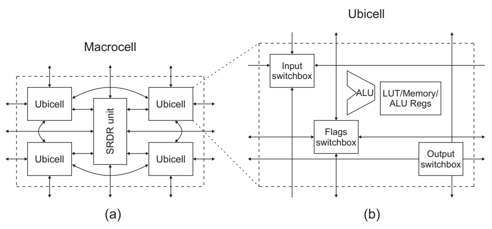

Fig. 2. Schematic organization of (a) Macrocell and (b) Ubicell (connection between ALU, LUT/Memory/ALU Regs section and other components are not shown).

Each Macrocell is composed of five blocks (Fig.2): SRDR responsible for the specific bio-inspired features like Self-Replication and Dynamic Routing (hence the name of the block – SRDR) [2], [3] and four Ubicells. In the FPGA mode, the Ubicell corresponds to a typical slice of the commercial FPGA device with four 4-input LUTs. The important feature of this mode is that all of the building blocks can implement both synchronous as well as purely asynchronous functions. On the other hand, in the SIMD mode the Ubicell acts as a 4-bit ALU with sixteen 4-bit registers [5]. What is worth noticing is that, the same memory elements building the LUTs in the FPGA mode are used as the mentioned registers in the SIMD mode.

The universal architecture of the Ubichip, together with the resource sharing, increase silicon area utilization but made the design more demanding as far as its physical implementation is concerned. The Ubicells may theoretically form asynchronous closed feedback loops which are the main obstacle to overcome in this project. Long connections between elements of the array of the Macrocells and the peripherals as well as rich connectivity patterns among the Ubicells and Macrocells drastically complicate the timing driven implementation process.

III. Theoretical Background

This section presents some details of the methods and models used during the proposed implementation scenario. Conclusion drawn from the process of performing Static Timing Analysis (STA) together with basic facts about timing models are used to target the implementation challenges resulting from the specific architecture described in the previous section. Additionally, it presents some general consideration of possible approaches and role of EDA tools in the implementation process.

A. *Static Timing Analysis*

In our methodology we use two basic techniques to verify whether our design meets timing requirements: timing simulation (with back-annotated delays) and Static Timing Analysis (STA). The principles of operation of both methods differ significantly and one can easily find their comprehensive description (for instance in [6]). Understanding the basic concept of STA is essential in the presented approach, since it is the basic tool in the process of design optimization.

The first important observation about the STA engine is that it performs timing checks on the entire design. It verifies timing requirements taking into account all possible combinations of logic connections including paths that cannot be established in a practical case (the substantial problem for the reconfigurable devices). Timing violations can be observed on paths resulted from mutually exclusive logic conditions or on unwanted paths constituting connections that make no sense from designer's point of view, like those exercising unused combinations of configuration bits. These false paths, regardless of their cause, should be excluded from STA. Otherwise both Quality of Result (QoR) and optimization time will be affected. Whereas the first mentioned type of false paths can be automatically detected by EDA tools (like Cadence Conformal Constraint Designer), the latter group requires designer's attention and careful consideration. Eliminating unwanted paths from timing analysis is a key issue during the implementation process presented in this paper.

Generally, STA is done in two main steps. In the first one the transition time observed at all nodes is calculated taking into account capacitance load. The STA engine tracks the timing path from the signal source to the sink. In the other stage, each cell delay is calculated individually taking into account the cell load and the calculated input transition time. Therefore, in order to exclude a particular path from STA, one needs to take care of both delay and slew propagation.

B. *Timing Models*

In our project we employed the most popular and widely used timing model for digital designs the Liberty format. This open source standard initially introduced by Synopsys has been significantly developed over last few years. Some syntax extensions like Effective Current Source Model (ECSM) [7] from Cadence have been introduced to address high nonlinearity in deep submicron or nanometric processes. However, nonlinear modelling is beyond the scope of this paper. In this section we limit the consideration to the concept of timing arcs and its representation.

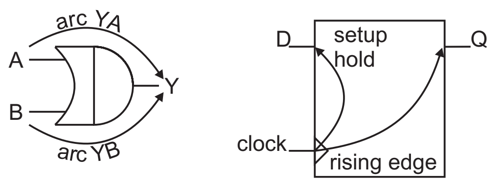

Fig. 3. Examples of combinational and sequential timing arcs.

The timing arc is a path during timing analysis that starts at an input, output, or I/O pin and ends at an output or I/O pin [8]. The only exceptions to that rule are hold and setup constraints (checks) – see Fig.3. In general, timing arcs fall into two main groups depending on their type: sequential (edge-sensitive) and asynchronous. This feature is very useful in process of specifying timing constraints presented later in this paper.

C. Flat versus Hierarchical Approach

Since we understand the implementation as a process starting from the RTL level and ending at a layer mask map, we present some trade-offs of both logic synthesis and place and route together.

The implementation of each system can be done using different approaches. One of them, called flat, is based on global synthesis without hard block boundaries during place and route. The flat approach enables global optimization techniques. What is more, avoiding hard hierarchical boundaries during the place and route process leads to better silicon utilisation (there are no placement halos, routing channels, etc.). Finally, it is possible to perform comprehensive full-chip STA which directly specifies the chip performance.

One the other hand, this general consideration does not have to apply to all designs. The PERPLEXUS chip, containing repeated logic blocks, seems to be an ideal candidate for the hierarchical bottom-up approach. However, there are some drawbacks of this method. As an example, it is extremely sensitive to quality of a floorplan. Moreover, routing within different hierarchical levels does not share the same metal layers. All this limits the silicon utilisation.

The EDA tools capacity is also an important aspect of choosing the implementation strategy. Obviously, there has been a great progress in EDA solutions over past few years. New software engines (using multi-threading) and sophisticated algorithms make it possible to obtain chips with multi-million gates. However, the Ubichip shows that gate count is not the only obstacle for the flat implementation process. Elimination of timing loops, related to ability to effectively set timing constraints (requirements), is the key problem which is described in more detail in the next sections.

D. Combinational Timing Loops

As it was mentioned before, each Macrocell in the array implements sequential as well as purely combinational functionality. From the timing point of view, it means numerous paths creating so called combinational (feedback) timing loops. The building blocks of the Macrocells do not have all output registered. In other words, the Ubicells and SRDR blocks can be seen as purely combinational (transparent) cells. This cause of action is seen as bad coding practice by some authors (see [9]). However, in case of reconfigurable devices this transparency is introduced on purpose. As a consequence, signal can theoretically traverse many configurable blocks and form a closed path. What is more, the number of closed paths rapidly grows with the number of reconfigurable blocks – in case of PERPLEXUS – size of the array.

A combinational loop (Fig.4) is a cyclic path made of timing arcs. The dependence of the combinational gate output and its input prevents from performing STA, since the engine cannot track the signal endpoints. Although there were new algorithms developed to analyse circuits with timing loops [10], [11] in some situations, this phenomenon should be avoided in standard implementation flows. The way the EDA tools deal with detecting and eliminating timing loops may differ even among tools from the same vendor (in our case Cadence Design Systems). As a common feature, the presence of timing loops results in a warning message and requires further investigation. In case of devices like the Ubichip dealing with this issue is essential. To illustrate the scale of the problem let us bring some numbers from reports after assembling the final design in Cadence Encounter Digital Implementation System. It occurred that the design, with approximately one million instances, includes more than 41 thousands timing loops.

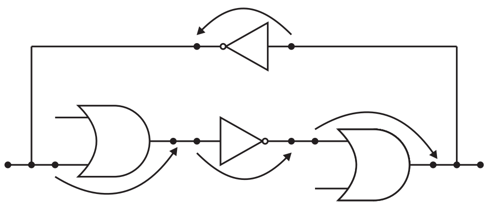
Fig. 4. Combinational timing loop with component timing arc.

It is worth pointing out that the presence of combinational timing loops in case of "standard" digital devices suggests problems with bad coding or poor architecture design. However in case of the reconfigurable devices, this phenomenon is unavoidable. By its nature, this class of devices gives a great number of possible data paths. It is the user who specifies the final datapaths in the design during the configuration process. It means that before loading a particular bit-stream which sets the connection pattern, the reconfigurable design contains numerous unpractical and undesired timing paths forming closed loops. The amount of these loops grows with the number of reconfigurable blocks in the design

Let us illustrate this issue employing some numbers from the Ubichip implementation process. The SRDR unit is built from more than 1300 instances and has nearly 280 ports. Each Ubicell unit can be characterized by almost 1900 instances and more than 170 ports. Obviously not all ports are combinational ("transparent"). However, when we take into consideration that we have 500 instances of SRDR and Ubicell blocks in a singe die, we end up with very high number internal ports and interconnections. Thus the number of timing loops is of the order of thousands.

The number of timing loops and the gate count made the global analysis and synthesis of the flat netlist simply impossible. As a consequence, we decided to develop a hierarchical bottom-up flow during logic synthesis.

IV. Implementation Flow

In contrast to traditional flows we do not use the front-end term since we are performing P&R session to get the design netlist. Therefore, we divided the implementation path into three main steps: getting the netlist, getting the layout, and performing timing validation.

A. Step One – Getting the Netlist

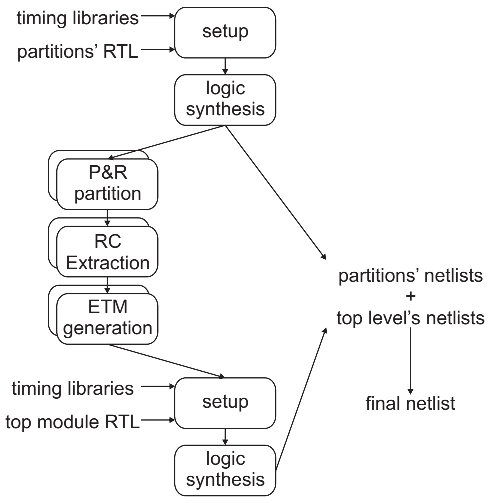

Fig. 5. First step of the implementation flow.

Fig.5 depicts the first step of the proposed flow aimed to create the netlist. As it can be seen in the figure, the first step involves both logic synthesis and the place and route sessions.

Obviously this step requires appropriate RTL description. First of all, the Ubichip RTL code needs to be divided into separate files for each element of the Macrocell Array and the top module. Specification of the blocks, called partitions, requires careful consideration. It should be done in order to balance efforts on setting timing constraints and placing and routing the design. In other words, too small partitions (grains) result in a poor floorplan with many placement and routing halos. On the other hand, relatively big logic blocks with great number of combinational paths are difficult to constrain.

Initially, the partitions are seen as empty modules at the top level. Clearly, the logic synthesis gives more reliable results if timing information for all blocks is available. The tools should take into account setup and hold timing checks to try to optimise the glue logic on the top level. Also area occupied by each partition is necessary to estimate delays on interconnections before the full placement is available.

As a consequence, the blackboxes have to be replaced by the timing representation of the partitions as soon as possible. Therefore, our main goal at this stage is to gather as much timing information about the partitions as possible for the top-level synthesis. At the same time we need to avoid undesirable timing loops. The solution of this problem involves creating a reusable representation of partition blocks – Extracting Timing Models (ETMs) [12].

1) Extracting Timing Model: Fig.6 shows a simplified process of creating an ETM which can be seen as a type of encapsulation. A system made of logic cells is transformed into a set of timing arcs. The tool can merge timing arcs of all components using the STA engine and create one cell. Netlist, extracted parasitic data and timing libraries are required for creating an ETM. Thus, the additional P&R process for each partition is needed. Timing context of a cell derived using the ETM is then used in the top level logic synthesis. It has to be taken into account, that this modelling introduces some

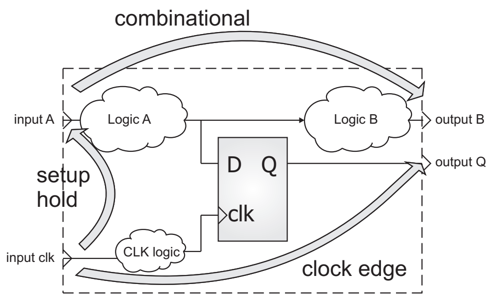

Fig. 6. First step of the implementation flow.

level of inaccuracy due to the tool limitations [12]. Precision of the model is also reduced by the lack of information on the final physical aspects of the block (including pin positions).

2) Timing loops elimination: Dividing the flow into syntheses of individual partitions simplifies process of specifying timing requirements – problem of constraining the design is decomposed into several synthesis processes resulting in a tighter control over the partition timing. However, such an approach does not solve the problem of avoiding timing loops at the top level of the design. There are several solutions for dealing with timing loops. In this section, we present some of them.

The first technique we investigated was using the `set_false_path` command from the Synopsys Design Constraints (SDC) set. This command removes timing constraints from a specified path. SDC is the most popular and widely-used format of representing timing intent. It is based on TCL language making it more flexible. However, it applies only to the propagation delay. Taking into account the information presented earlier in this document, STA would compute the path since slew rates are still propagated.

The next technique is so called multi-mode synthesis. In this solution timing analysis of the design is done complying with a set of logical values applied to the design. In fact, we were successfully using this method with setting the timing requirements. One of the Ubicell sections was constrained differently for each mode to follow a varying logic path length according to the current configuration. Despite of its clear usefulness in case of the Ubicell, this method could not be applied to the top level synthesis in order to eliminate timing loops. We encountered several barriers while trying to exercise this method. First of all, maintaining satisfactory control over the timing, without decomposing the design into smaller blocks, in this case would be very demanding. Despite difficulties in creation of timing constraints for each mode itself, there is also the synthesis tool capacity issue. In order to prevent slew to be propagated through undesirable paths, each mode would require a separate timing graph during optimization. By the time of the PERPLEXUS project realisation (2009/2010) this feature was not available. What is more, it is said that creating a new timing graph results in increase in synthesis time for more than 40 per cent. Assuming four modes (which is a relatively small number for a system of this level of complexity) and about one million instances, impact on synthesis effectiveness cannot be neglected.

Finally, after checking several possibilities we decided to avoid the unwanted input-output timing arcs of each partition using the SDC `set_disable_timing` command which literally cuts the timing paths. However, since this solution is considered to be relatively unsafe (it removes paths entirely from timing reports), we needed to ensure that only already optimized paths that create the problem of timing loops are disabled.

3) Discussion: Combination of the timing models and the timing arc disabling brings several profits to the implementation process. On the other hand, it has to be taken into account that the mentioned limitations of the models and disabling timing arcs results in the need of extra efforts and attention during specifying timing constraints. In other words, the risk of losing control over the timing of the design increases dramatically. Some paths can be excluded from timing optimization and reports by mistake. All this implies the necessity of performing the comprehensive full-chip timing validation which will be covered in one of the further sections of this paper.

B. Step Two – Getting the Final Layout

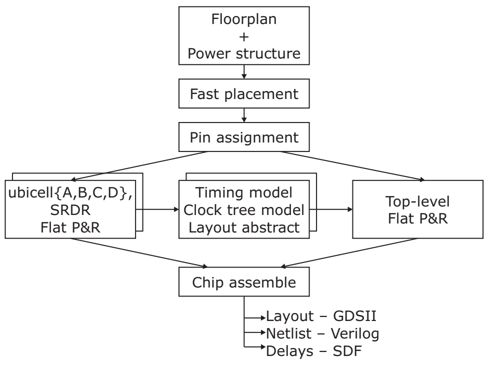

Fig. 7. Block diagram of the step two.

Fig.7 shows the successive stages of the process of getting the final layout using Cadence Encounter Digital Implementation System. This step can be seen as more conventional in terms of standard implementation flows. However, the methodology presented in this paper can be considered as a combination or a hybrid of top-down and bottom-up approaches. In the first case, all partitions are placed and timing requirements are derived from the top SDC file. We do not perform time budgeting and the optimization process is based on constraints developed for partitions and top module separately. From the other side, we are creating a top level floorplan and power structure, as in the top-down approach, to better use the available area.

Another important distinction from a standard place and route process is the fact that we are employing the partition cloning technique. In this method one block (referred as a master partition) is chosen and instantiated (cloned) multiple times. Only the master block is optimised and changes to individual clones are not possible. As a consequence a clock signal propagation network (clock tree) cannot be implemented in a single ”flat” session [9], [13]. This section presents how this and other aspects related to the partition cloning are addressed in our flow.

FPGA devices are known of their high metal density [14]. Therefore, the design floorplan must allow intensive intra-chip connections. Specifying the partition positions and routing channels is the biggest challenge in this step. The key issue of planning the floor is the trade-off between an area allocated for global (top-level) connections and local routing inside the partitions. The presence of master partitions and their duplicates additionally complicates the planning the floor, since each clone needs to be aligned to power structure identically to avoid power and ground shorts.

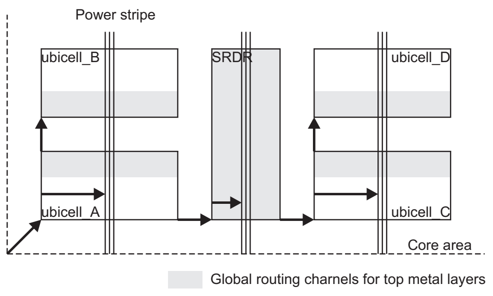

Fig. 8. Relative floor and power plan for Ubichip.

Fig.8 depicts the proposed approach to creating the floorplan for the Ubichip. Positions of partition boundaries and power stripes are specified relatively which is illustrated using arrows. The combination of TCL commands creates a set of related numbers like distance between partitions, routing channel widths or position of power stripes over the partition. All sizes are generated automatically after supplying several basic figures (like aspect ratio and size of partitions) and rounded to placement/routing grid. This automation is especially important taking into account that achieving trade-off between local and global routing may require several iterations.

In order to obtain the most optimized floorplan we considered several possible solutions for routing channel widths, block sizes, etc. It has to be pointed out that all the changes had to follow the requirements set by the power plan. For instance all partitions needed to be located on the power-ground structure in the same way. If we take into account that the power horizontal rails are routed in the ground-supply-ground pattern, the smallest possible adjustment to the vertical routing channel widths was two rows.

Our objective was to get the final floorplan as soon as possible avoiding many iterations of the entire place and route process. To achieve that goal we decided to use trial route option. This routing estimation can give an idea of possible congestion areas without performing the complete place and route. Obviously not all problems could be foreseen at the floorplan level. Nevertheless, the well-structured and firmly divided flow facilitates finding source of possible problems and adjusting the design decisions to avoid them.

Partition cloning is not a widely-used technique even with multi-core processors. It results from the fact that performance of each logic block depends heavily on its position on a silicon die. Aspects like connection lengths, neighbourhood, etc. play here an important role. What is more, the number of repeated blocks in processors available on market is relatively small. In case of the PERPLEXUS project the situation was quite different. We were implementing a chip with a few hundreds of repeated partitions. In our approach we tried to balance high number of partitions with impact of its physical location and connections.

As it was mentioned, each of four Ubicells building the Macrocell has the same behavioural description. However, duplicating the same layout for every partition would severely limit the optimization capabilities. For instance, fixed pin locations increases wire length which affects timing. As a consequence, the final netlist was changed in order to address this problem. The netlist of the Ubicell was copied and renamed in order to obtain four different blocks instantiated according to their physical position. This way, functionally identical blocks were represented by four types of partitions (see Fig.8). This example shows again the role of careful organization of the design. The differentiation was needed only in the P&R process. Thus, the changes were done to the netlist, leaving the VHDL models verification process unaffected.

Choosing the master Macrocell was another problem to solve. The selection of physical part of the chip to be duplicated needs to be done carefully since it may significantly affect performance of the entire design. The master partition has to be optimized with realistic design constraints (mainly pin assignment). These constraints need to be the most representative for all partitions. Therefore, we considered two major factors while selecting the master partition: the partition neighbourhood and its distance to the relevant peripherals.

The next problem that arises here is the partition pin placement. Each partition contains a considerable large number of pins. Their distribution may noticeably affect the overall timing and requires careful attention. In the proposed methodology, the final pin positions are reached in the initial placement process. No timing optimization is done here in order to gain speed. In other words, this action can be seen as a trial placement. Pin location constraints can be applied as an enhancement of the process (in terms of QoR and speed). In case of the Ubichip it was based on specifying side and metal layer for the pins of particular block. Pin assignment is the last part of planning the floor before dividing the design into partitions.

It is worth emphasising that in case of partition cloning aspects like partition pin placement or careful power structure become even bigger challenge. Any possible errors (not optimal solutions) are repeated and its impact on the final QoR is significantly increased.

The final P&R of each partition does not differ from standard procedures known from conventional implementation flows. Instead of presenting it in detail we will concentrate on one aspect. After P&R for each partition (excluding the top module) we were generating a complete representation of the partitions including:

- layout abstract (LEF) including information on process antenna ratios for all pins
- timing model (ETM)
- clock tree macromodel including delays on internal clock tree

Unlike the ETM generated in the step one of the flow, the current model reflects timing of the partition more precisely. It takes into account the final placement of the blocks. Also latencies of the internal clock tree structures are fully represented in a clock tree macromodel. This way we obtain a fully reusable partition representation (macro). Consequently, each partition can be used as a leaf cell during P&R of the top module. Hierarchical clock tree synthesis is possible since the tool is aware of latencies in clock signal propagation inside and between macro blocks.

After chip assembling we obtain the final netlist and corresponding Standard Delay Format (SDF) [15] file which are used during timing verification of the design.

C. *Step Three – Timing Validation*

As it was mentioned before, we use two timing verification techniques: STA and back-annotated simulations. Since the designed circuit has a reconfigurable architecture, its timing can be verified only with respect to the current operation mode. Additionally, the timing arcs, which are removed in order to disable combinational loops, severely limit usability of STA to determine the actual top frequency of the circuit even in a given mode. As a consequence, timing simulations are introduced to the flow as its important element. Exhaustive tests and timing debugging can highlight hot spots in the design performance which can be then used to readjust the timing requirements set during the optimization process.

Using simulation tools like ModelSim with a standard approach (called direct simulations) is associated with limited bandwidth. We enhanced the process of verification by making tests more comprehensive to cover more possible configurations and simultaneously shrink the verification time. Therefore, we employed more advanced verification technique called constrained-random testbenches using Hardware Verification Language (HVL) with the VERA environment.

The individual parts of the Ubichip were represented at higher level of abstraction using HVL. The verification was based on automatic comparison of the responses of the HVL model and the final netlist upon randomly generated input vectors. After introducing a delay file and using appropriate models of the cell, we were able to judge the design timing. An SDF file includes delays as well as timing checks (setup, hold) [15] corresponding to Verilog specify block of the cell models. To illustrate the proposed timing validation let us assume that the clock period is too small for the signal to propagate between registers. As a consequence, the result becomes either wrong (different from expected) or unknown (containing Xs). In both cases it can be reported. Moreover, the simulations can be easily automated using TCL scripts. So, the top clock frequency for a given operation mode can be established after a set of simulation runs.

Random testbenches are typically used during validating behavioural models. The VHDL or Verilog model is considered to be correct when it gives the same results as the reference description. We used the same idea but applied it to the final netlist enriched with delay information. This approach proved to be extremely efficient due to the possibility of high-level modelling, input randomisation, and automatic result mismatch detection. As a result, we were able to automate the simulations and effectively use them together with the SDF file to determine the design timing. The mixture of constrained-random post-synthesis simulations and STA allowed us to overcome limitations of both verification methods: preserve close control over timing intent of the blocks and at the same time effectively validate performance of the entire design.

V. Flow Validation

Finally, the design was fabricated using the UMC 180 nm technology. A dedicated PCB test-board was created in order to check the simulation results with the physical measurements done with logic analyser. It allowed us to measure the top frequency of operation in different modes of operations. General Purpose IOs were used to observe internal signals in order to test the FPGA mode. Also the SIMD mode operation was successfully verified.

We obtained a very good correlation between foreseen and measured results. Additionally, correctness of the flow was confirmed with demonstration of bio-inspired algorithms that took place during the PERPLEXUS Final Review Meeting (29 March 2010).

VI. Conclusion

In our work we developed a consistent and complete implementation flow. As it was shown each partition is implemented as fully reusable macro. This way the high scalability of the architecture developed in the PERPLEXUS project was also reflected during the physical realisation phase. Impact of the size of the Macrocell Array on synthesis or place and route time was minimised due to multiple instantiation (cloning).

The project was also interesting from the management point of view. Most of the steps of the flow were done by the same group of people starting from adjusting the architecture and partitioning the design ending at timing validation. It is worth pointing out that successful stages of the presented methodology heavily depended on one another. To illustrate this situation let us consider the timing validation. The functional simulations put additional stress on the parts that were excluded from STA. Thus, the testbenches needed to somewhat take into consideration the timing constraints. The presented approach resembles to some extent term construct by correction introduced by the Sun Microsystems engineers that worked on the UltraSPARC processor [13]. The PERPLEXUS project also showed that all physical and design implication cannot be predicted at the architecture specification stage. Therefore, close collaboration within a working group is essential.

The presented methodology can be extended from the tools point of view. First of all, new features like virtual silicon prototyping are now available. In this method influence of parasitic elements can be taken into account during synthesis. The early estimation of parasitic elements may play an important role in designs where a complex floorplan substantially increases lengths of individual connection length. The presented flow may be also enriched with new more efficient methods of specifying timing constraints. It is well-know phenomena that timing intent of a design is hardly ever stated mature enough. A dedicated tool taking into account the specific architecture class may result in a more automated process of setting timing constraints.

References

[1] E. Sanchez, A. Perez-Uribe, A. Upegui, Y. Thoma, J. Moreno, A. Villa, H. Volken, A. Napieralski, G. Sassatelli, and E. Lawarec, “PERPLEXUS: Pervasive computing framework for modeling complex virtually-unbounded systems,” in *Proc. 2007 NASA/ESA Conference on Adaptive Hardware and Systems*, Edinburgh, UK, Aug. 2007, pp. 587–591.

[2] A. Upegui, Y. Thoma, A. Perez-Uribe, and E. Sanchez, “Dynamic routing on the ubichip: Toward synaptogenetic neural networks,” in *Proc. 2008 NASA/ESA Conference on Adaptive Hardware and Systems*, Noordwijk, The Netherlands, Aug. 2007, pp. 587–591.

[3] Y. Thoma, A. Upegui, A. Perez-Uribe, and E. Sanchez, “Self-replication mechanism by means of selfreconfiguration,” in *Proc. ARCS ’07 - 20th International Conference on Architecture of Computing Systems 2007*, Zurich, Switzerland, Mar. 2007.

[4] J. Moreno, J. Madrenas, and L. Kotynia, “Synchronous digital implementation of the AER communication scheme for emulating large-scale spiking neural networks models,” in *Proc. 2009 NASA/ESA Conference on Adaptive Hardware and Systems*, San Francisco, CA, Nov. 2009, pp. 189–196.

[5] J. Moreno and J. Madrenas, “A reconfigurable architecture for emulating large-scale bio-inspired systems,” in *Proc. IEEE Congress on Evolutionary Computation CEC 2009*, Trondheim, Norway, May 2009, pp. 126–133.

[6] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach.* Springer, 2009.

[7] “Si2 liberty syntax extensions for characterization and validation specification v1.0 21,” Silicon Integration Initiative, Inc., 2009.

[8] “Liberty user guide, vol. 1 version 2009.06,” Synopsys, Inc., 2009.

[9] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*, 3rd ed. Springer, 2007.

[10] T. R. Shiple and H. T. G. Berry, “Constructive analysis of cyclic circuits,” in *Proc. European Design and Test Conference (ED&TC) 1996*, Paris, France, Mar. 1996, pp. 328–333.

[11] A. Gupta and C. Selvidge, “Acyclic modeling of combinational loops,” in *Proc. IEEE/ACM International Conference on Computer-Aided Design ICCAD-2005*, San Jose, CA, May 2005, pp. 343–347.

[12] “Encounter digital implementation system user guide product version 9.1,” Cadence Design Systems, Inc., 2009.

[13] H.Bhatnagar, *Advanced ASIC Chip Synthesis Using Synopsys Design Compiler Physical Compiler and PrimeTime*, 2nd ed. Springer, 2001.

[14] P. Zuchowski, C. Reynolds, R. Grupp, S. Davis, B. Cremen, and B. Troxel, “Hybrid ASIC and FPGA architecture,” in *Proc. IEEE/ACM International Conference on Computer Aided Design ICCAD 2002*, San Jose, CA, Nov. 2002, pp. 187–194.

[15] “Standard delay format specification, version 3.0,” Open Verilog International, 1995.