Automated Physical Hierarchy Generation: Tools and Methodology

Ali El-Zein[1] Alvan Ng[1] Benedikt Geukes[2] Maya H. Safieddine[1] Wolfgang Roesner[1]

IBM Systems and Technology Group
[1]Austin, TX, United States
[2]Boeblingen, Germany

email: {elzein, alvan, wolfgang}@us.ibm.com
benedikt.geukes@de.ibm.com
maya.safieddine@ibm.com

***Abstract* - Creating a physical hierarchy from a logical hierarchy in complex chip designs requires months of back and forth negotiation between the frontend and backend teams. We present a novel hierarchy manipulation tool and methodology that enables automatic generation of the physical hierarchy resulting in easier front-end verification and better physical implementation. The transformation is sound as proven by equivalence checking using formal verification. We deployed the tool into a methodology that is currently used in an industrial setting wherein it reduced design bugs in certain areas by more than 50%.**

I. Introduction

Hardware designs are hierarchical in nature. Modularity is achieved by encapsulating certain logic functionality as a node in the design hierarchy. Logic designers and verification engineers tend to group logical blocks that are functionally related close to each other in the design hierarchy referred to as functional hierarchy. The functional hierarchy may not map well to a two-dimensional layout solution since it is usually conceived with little or no consideration of the layout information, such as area and routing constraints. Logic designers are forced to map their logic to a physical hierarchy where things are grouped according to geographic proximity. Strict enforcement of a single Functional/Physical hierarchy makes logic design and verification less efficient as placement of logically adjacent blocks along physical design geographies scatters adjacent logic into separated areas of the RTL hierarchy. Logic designers as owners of the HDL will be under constant pressure to modify their HDL due to physical constraints. So instead of a single hierarchy, tools are developed to automatically derive the physical hierarchy from the logical hierarchy [1].

However, such tools were not fully utilized in industry due to the following strict requirements.

- Physical constraints need to be preserved and controlled on the generated physical hierarchy. This includes pin cloning, feedthroughs, attribute propagation, etc.
- Simulation environments (checkers and drivers) needs to run on both hierarchies
- Efficient and scalable formal verification up to the chip level needs to exist to verify that both hierarchies are equivalent
- Name stability and predictable port name generation in the physical hierarchy are required
- Small changes on the input HDL should result in small changes in the output VHDL. This is very essential for Engineering Change Order (ECOs).

IBM Power 9 [2] is the first major chip inside IBM to use logical to physical hierarchy mapping on a large scale. The mapping targets three major areas:

1) exploring timing and area improvement,
2) reducing bugs in complex Reliability, Availability and Serviceability (RAS) infrastructure, referred to as pervasive logic, by isolating the pervasive design into a separate unit and automatically redistributing it throughout the chip, and
3) dynamic creation of physical chiplets, grouping of functional units, while preserving the original logical HDL hierarchy tailored to their verification environment.

Automating the logical to physical hierarchy mapping, while adhering to the aforementioned requirements, is essential to achieve these objectives. In the larger context, IBM recently made a big push toward Aspect (Concern)

Oriented Design [3-4] where different aspects (concerns) of the design, such as DFT, POWER, Trace Debug, etc., are developed separately and later weaved in to form the complete design. Automatically generating a physical hierarchy is an essential step in this effort.

In this paper, we introduce a hierarchy generation tool Morph-Hier that automates logical to physical hierarchy mapping and address its limitations in the design flow. The tool allows for major productivity boost, demonstrated in achieving all three targets of the IBM Power 9 hierarchy mapping. We explain how Morph-Hier is incorporated in the overall design methodology, with emphasis on:

- Structure of the tool input command files, which we also call *recipe files*
- When to run the tool in the design flow and how to formally verify results

The remainder of the paper is organized as follows. Section II describes the overall hierarchy manipulation tool and its building blocks. Section III discusses grouping small synthesizable blocks into larger blocks. Section IV describes building chiplets dynamically. Section V discusses pervasive logic and its distribution into the physical hierarchy. Section VI presents the experimental results. Finally, Section VII concludes the work.

II. MORPH-HIER: BUILDING BLOCKS

Automatically generating a physical hierarchy from a logical hierarchy is not a new concept. It has been around for decades. All previous attempts, however, failed as the focus was mainly on moving an instance from a source to a destination, creating and removing ports along the way. Although this is a base block, additional supporting operations are required for a functional hierarchy manipulation tool. We built Morph-Hier based on generic building blocks, which can be used to create larger hierarchy manipulation scenarios.

The main building blocks of Morph-HIER are (1) *Wrapper Creation and Deletion*, (2) *Instance Move*, (3) *Flattening Instance/Entity*, (4) *Dealing with Signals (ports and internal signals)*, (5) *Ports Optimization* including *Port Clone* and *Naming Mechanism* for pins, nets and instances as two sub blocks, (6) *Throw and Catch*, (7) *Feedthroughs Creation* by inserting buffer or pair of inverters, (8) *Component Clone*, (9) *Flexibility of Recipe File*, (10) *Attribute Propagation*, (11) *Equivalency Checking*, and (12) *Name Mapping Database.*

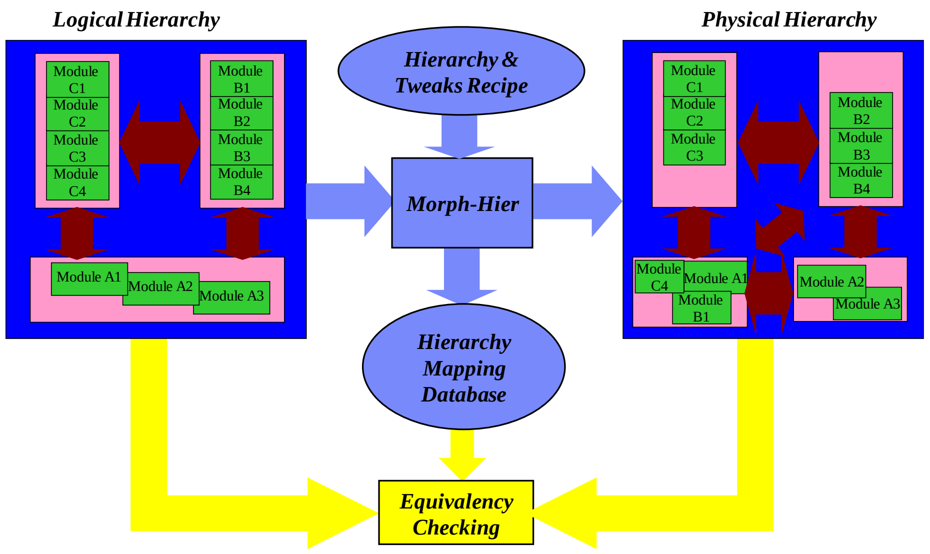


Figure 1. Overview of Morph-Hier flow.

Morph-Hier supports the operations of these building blocks for both the input logical/functional hierarchy and the output physical hierarchy. For example, creating feedthroughs operation targets mainly the physical hierarchy.

Fig. 1 presents an overview of the Morph-Hier flow. The tool takes as input a logical hierarchical design and a set of commands in a recipe file. It updates the design by moving instances, creating wrappers, and applying necessary operations specified by the recipe file. It generates a physical design hierarchy along with a hierarchy mapping database. The database allows mapping of hierarchical names of design elements that have different places in both hierarchies. The input and output designs are verified for equivalency by running equivalency checking algorithms.

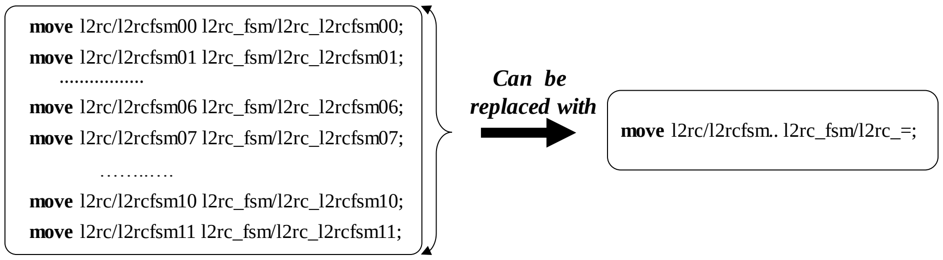
In the rest of this section, we detail recipe files, moving instances, cloning ports, and creating feedthroughs as main operations of the tool. We first introduce recipe files as essential tool input to determine the required manipulation.

A. *Recipe Files*

Recipe files contain a set of commands for hierarchy manipulation. A recipe file is attached to a design entity. All hierarchical references inside the recipe file are rooted with the design entity they are attached to. The tool applies recipe files starting from the lowest levels of the hierarchy and works upward from there. Recipe file commands are declarative and not sequential [5]. Morph-Hier will order the execution of the command to yield consistent results. The applied operations are of the following order:

- Wrapper creations
- Wrapper instantiations
- Clone instantiations
- Pre-move hierarchy flattening statements
- Move statements, which are ordered to yield consistent results
- Post-move hierarchy flattening statements

Recipe files also support regular expressions and pattern substitution to minimize it size as illustrated in Fig. 2.



```
move l2rc/l2rcfsm00 l2rc_fsm/l2rc_l2rcfsm00;
move l2rc/l2rcfsm01 l2rc_fsm/l2rc_l2rcfsm01;
    .................
move l2rc/l2rcfsm06 l2rc_fsm/l2rc_l2rcfsm06;
move l2rc/l2rcfsm07 l2rc_fsm/l2rc_l2rcfsm07;
          ……..….
move l2rc/l2rcfsm10 l2rc_fsm/l2rc_l2rcfsm10;
move l2rc/l2rcfsm11 l2rc_fsm/l2rc_l2rcfsm11;
```

```
move l2rc/l2rcfsm.. l2rc_fsm/l2rc_=;
```

Figure 2. Pattern expansion and regular expression flexibility of recipe file.

B. *Moving an Instance*

Moving an instance from a source to a destination involves the following steps.

1) Creating optimized port lists on all levels of the affected hierarchy. If a signal connection to a port in the source location is already available at the destination, no ports will be created. This is not always desirable on the physical hierarchy due to wiring constraints. In such cases, pin cloning will be explicitly specified as will be described in section II.C.
2) For the cases where moves (whether from or to) effect an entity that is instantiated multiple times, the tool figures out if the same components and their connectivity are moved in or out of all entity instances in the same way. If this is the case, then the affected entity will remain intact; otherwise different entities will be created for different configurations.
3) Most complicated IPs are designed bottom up. This means that any new VHDL wrappers that are created by Morph-Hier should be identical irrespective of the level of hierarchy at which the tool is run. An Audit tool was developed in order to ensure equivalency of such wrappers.

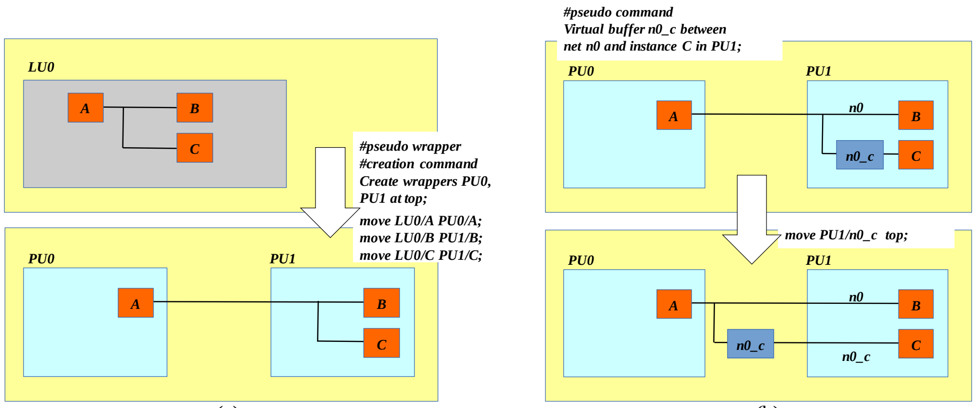
C. *Cloning ports and Creating Feedthroughs*

For cases where port optimization is not desired due to wiring or timing constraints, Morph-Hier duplicates pins upon moving an instance. A common construct based on *virtual buffers* is used for both duplicating pins and creating feedthroughs.

Fig. 3(a) shows an example where ports are optimized. Fig. 3(b) shows how creating a virtual buffer can disable port optimization. Instead of creating a special pin cloning command, pins are cloned as part of the buffer creation and move. This highly simplifies the recipe file syntax. It is a two-step process. First a virtual buffer is inserted at the desired cut point. Then, by moving the virtual buffer to the desired location, port optimization will be disabled.

The virtual buffer will later be dissolved by the tool resulting in pins being cloned. As can be seen, pin cloning recipe commands operate on the physical hierarchy.

The same concept can be used to create feedthroughs. Fig. 4 shows an example of physical hierarchy creation. Fig. 5 shows how to route a signal at the top level through another component by creating and moving a virtual buffer.



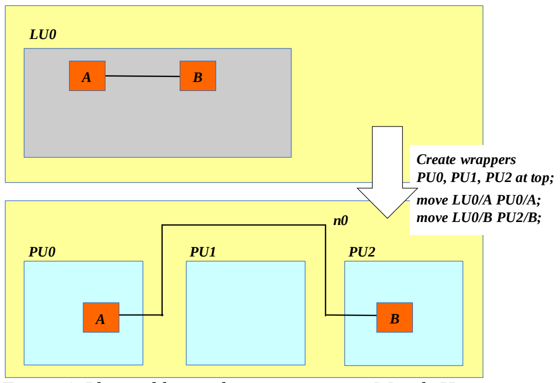
```
#pseudo wrapper
#creation command
Create wrappers PU0,
PU1 at top;
move LU0/A PU0/A;
move LU0/B PU1/B;
move LU0/C PU1/C;
```

```
#pseudo command
Virtual buffer n0_c between
net n0 and instance C in PU1;
```

```
move PU1/n0_c top;
```

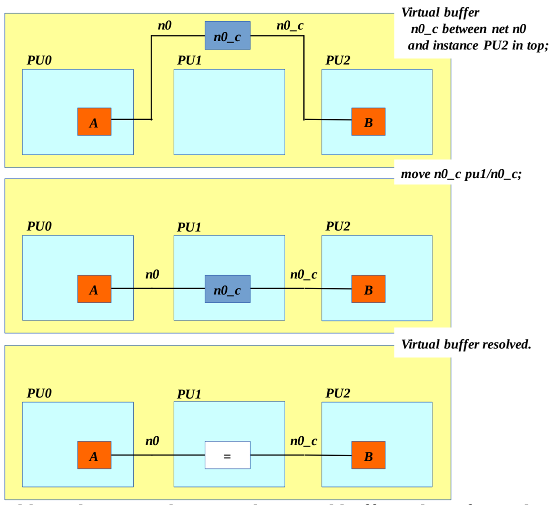
(a) (b)

Figure 3. Instance move (a) with port optimization and (b) without port optimization.



```
Create wrappers
PU0, PU1, PU2 at top;
move LU0/A PU0/A;
move LU0/B PU2/B;
```

Figure 4. Physical hierarchy creation using Morph-Hier.



```
Virtual buffer
 n0_c between net n0
 and instance PU2 in top;
```

```
move n0_c pu1/n0_c;
```

```
Virtual buffer resolved.
```

Figure 5. Feedthrough creation by using the virtual buffer utility of Morph-Hier.

III. LARGE SYNTHESIZABLE BLOCK CREATION

One popular application of Morph-Hier is to group small logic blocks together to create a large physical block for synthesis. Logic designers will code VHDL entities based on the smallest functional granularity and reusability, e.g. state machine, arbiter, queue, and pipe line stage, etc. An example of such a design is shown in Fig. 6. The design is naturally broken down into three functional modules: interface handler, arbiter, and dataflow. It is apparent that such functional modules can be owned by different designers and verified independently. It adheres to the principle of design modularity.

Functional hierarchy usually does not lend itself to the best physical implementation. The design team will take it through the physical design planning process. Logical wrappers are dissolved and the physical affinity of the small logic blocks are examined. Based on micro-architecture requirements, timing criticality, wire congestion, and other physical constraints, a floorplan is formed by placing the small logic blocks. An example of such a floorplan is shown in Fig. 7.

The next step is to determine which logic blocks should be grouped together to form a large block for synthesis. The decision is based on physical affinity, taking into account factors like synthesis processing time, area efficiency, circuit and interconnect optimization, criticality and logic stability, etc. It has been demonstrated that a large block can yield better synthesis optimization. It can achieve a compact floorplan with higher overall area utilization as well as better timing and lower power. Hence, it is desirable to create a large block whenever the implementation risks can be balanced.

Fig. 8 depicts such a physical implementation. The large physical entities are created using IBM's Morph-Hier tool. It makes this kind of logical to physical transformation extremely easy and fast. The turnaround time is on the order of hours versus days without the need to circle back to the logic design team. A large part of the designer's time is spent on coding up recipe files to specify the creation of physical blocks and their grouping. The actual run time of Morph-Hier takes only minutes. Alternate grouping can be explored quickly by simply changing the recipe files and rerunning Morph-Hier. This methodology brings large productivity gain to the IBM Power9 design team as it enables the design team to explore different optimization options quickly.

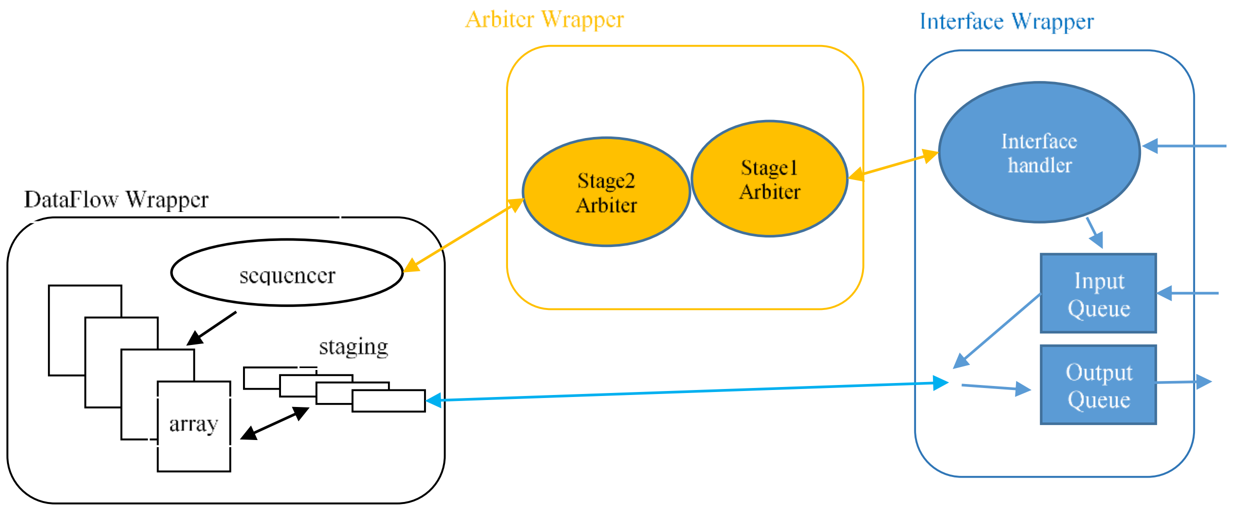


Figure 6. Logical VHDL hierarchy.

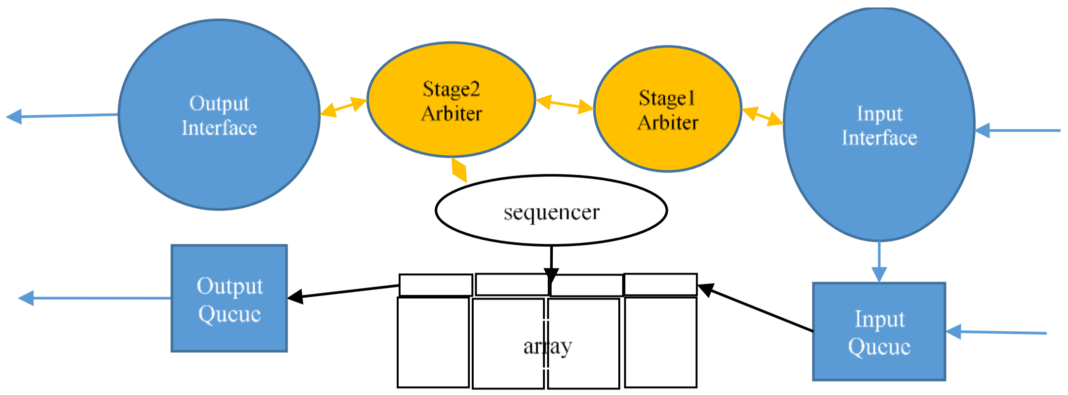


Figure 7. Physical floorplan with small logic blocks placed.

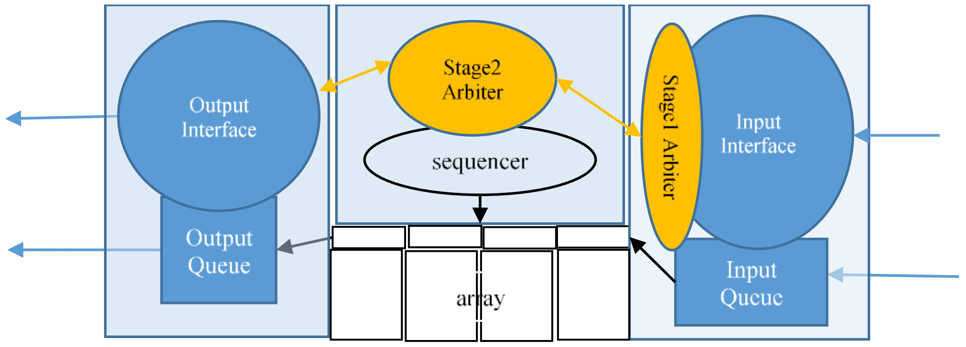


Figure 8. Physical floorplan with large synthesizable blocks.

IV. Chiplet Creation

The benefit of hierarchy transformation is more apparent when applied at the chip level. Today's complex multi-core processors are packed with an increasing number of processor cores plus numerous types of on-chip accelerators and peripherals [6]. Optimally integrating such designs is challenging and requires detailed physical design planning, especially when chip area is nearing the die size limit. Since logic entry and verification often start before actual physical floorplanning takes place, logical organization is the preferred way to build a chip VHDL. It helps design teams to focus on developing functionality. Fig. 9 shows an example of a common chip VHDL hierarchy organized logically. An on-chip bus or interconnect defines a standard interface to connect all the components. Miscellaneous control circuitries are also spread all over the chip.

In the early stage of a project, physical design teams have little content to work off except for the estimated box sizes of each functional unit to form the initial concept of the chip floorplan. Chiplets are defined based on integration granularity, design modularity, and physical area. However, leading edge chip design is an ever-changing endeavor. Disruptions like new requirements, technology short falls, and incorrect initial sizing could reset the floorplan and result in the need to redraw chiplet boundaries. With conventional manual method, the chip and chiplet VHDLs must be recoded to reflect the new chiplet groupings. This could take weeks for the changes to ripple through the design database and verification model.

IBM Power 9 design team undertook an innovative path deploying Morph-Hier to perform hierarchy manipulation to create all the chiplets automatically. It provides a cleaner and easier way to reconfigure the chip's physical hierarchy while leaving the functional hierarchy intact and avoiding impact on verification. The team has successfully created multiple revisions of the chip floorplans with various changes in the chiplet grouping using Morph-Hier. Derivative chips are also in the work and Morph-Hier has demonstrated its value in speeding up chip development by keeping the functional hierarchy independent. Fig. 10 and Fig. 11 show a couple examples of the chip floorplan to illustrate Morph-Hier's ability to generate different complex physical hierarchies at large scale.

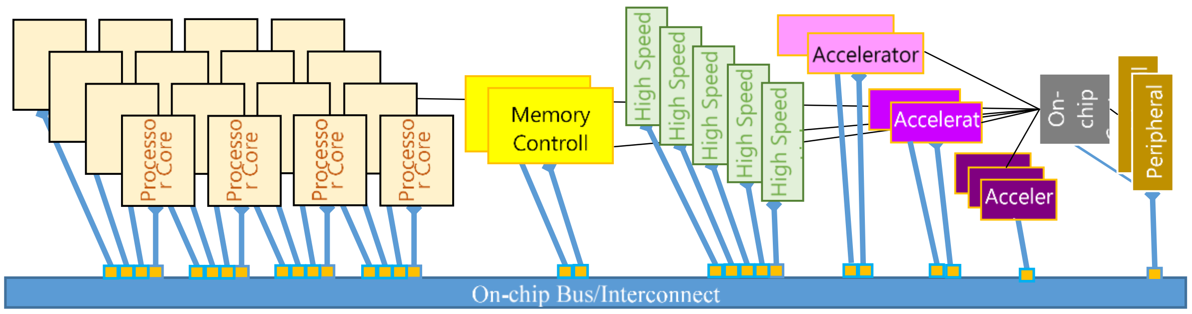


Figure 9. Multi-core processor chip logical organization.

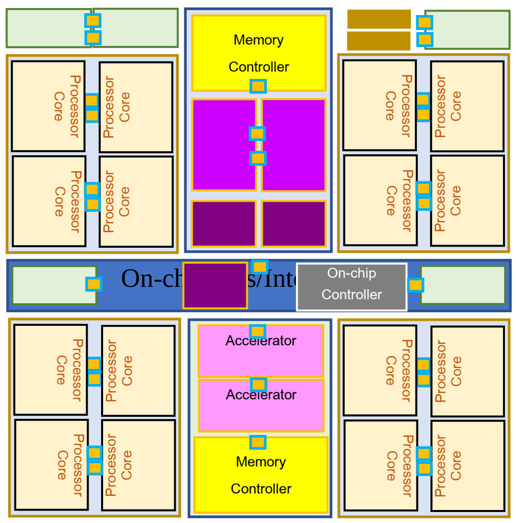


Figure 10. Implementation 1:

- Quad-core chiplet instantiated 4 times
- Center stripe bus chiplet with 2x high speed link, 1 small accelerator, and the on-chip controller
- Top chiplet contains 1 memory controller, 2 small accelerators, and 2 medium accelerators
- Bottom chiplet contains 1 memory controller and 2 large accelerators

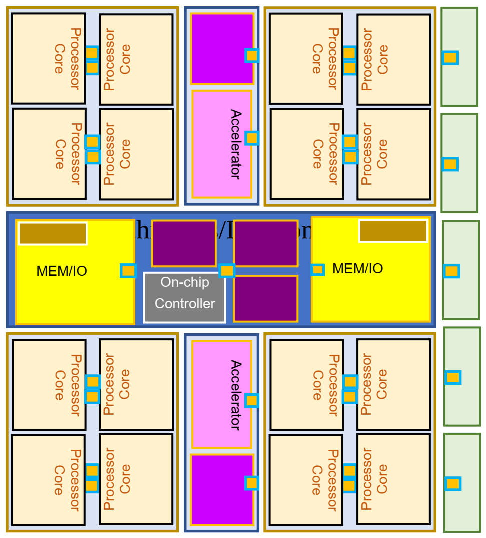


Figure 11. Implementation 2:

- Quad-core chiplet instantiated 4 times
- Center stripe bus chiplet contains 2x Memory-Peripheral combined unit, 3x small accelerator, and the on-chip controller
- One accelerator chiplet instantiated twice which contains a large and a medium accelerator

V. Pervasive Application

Besides utilizing Morph-Hier with a focus on driving an optimized physical design or integration solution, the tool enabled simplified design and verification mechanics in specific areas of a complex microprocessor architecture.

In addition to typical functional units, like a core unit, memory controller units, accelerator units, IBM High Performance Micro Processors contain a variety of pervasive logic blocks. Key contributors to these blocks are clock control, test and debug logic. By nature, these functions are widely distributed over the complete physical chip area and at the same time interact with every functional unit on the chip.

In the traditional/classical design approach, each functional unit pulls in small dedicated parts of the overall pervasive architecture for the target localized pervasive functionality as well as the network to interconnect those dedicated pervasive logic blocks with the main pervasive control instance. With this integrated approach functional units not only have to have fundamental knowledge of the pervasive functionality itself but also need to incorporate non-functional design aspects into their logic and physical design activities. In addition, full functional verification of the pervasive components and interconnects is only possible on a full chip view, once all components are available and interconnected to each other. Such a full chip view, usually is available much later in a project's time schedule then dedicated units. As a result, parallel and independent design and verification approaches are difficult to implement due to the high number of cross dependencies throughout the various disciplines. However, such independent design and verification approaches (a separation between functional design and pervasive design) would be possible while considering the logic design aspects only and skipping at first over the physical ones. The introduction of morph hierarchy enabled such a separation and reduced the interaction of pervasive logic design with a "mainline" logic design to a minimum. Introducing such a separation comes along with a standardized interface as handover point between mainline unit and pervasive logic design. Having mainline unit and pervasive design unit less cross-coupled opens a lot of new opportunities in speeding up of the overall design cycle:

1) *Parallel logic Design*
Functional units can start designing almost independently from pervasive logic design. A functional unit is designing towards the standardized interface. The pervasive logic design team almost only needs to know the number of functional units requiring such a standardized interface. Then the pervasive logic design team can start implementing the pervasive network and main control logic.
2) *"Plug and Play" of units*
With the standardized interface, units become swappable from a pervasive logic design perspective. Custom solutions for dedicated units are reduced to a minimum.
3) *Verification Speedup*
Speedup in verification is achieved in multiple areas. As the pervasive logic design functionally is now separated from the mainline units, its logic design is moved into a dedicated pervasive logic design hierarchy, which is like a chip scope, but not dependent on it. With this, verification of pervasive logic can start right away with the availability of the pervasive logic. Dependencies on having functional designer implemented pervasive logic is resolved and reduced to the implementation of the standardized unit interface.
Functional units can start verification independent of the complete pervasive logic design availability. A functional unit verification is working towards the standardized unit interface.
4) *Design Quality*
As functional designer can now focus on functional design, the number of bugs in the pervasive logic design resided within the functional units have significantly decreased.

Fig. 12 shows the separation of pervasive logic from functional logic enabled by introducing Morph-Hier to the design flow. Fig. 13 shows the target design view having functional design and pervasive design closely interlocked and embedded. The pervasive logic content and its connections, simplified and shown in Fig. 12 as the red box/dots in the red circle, is now distributed across the complete chip. Instead of coding the pervasive infrastructure in the view of Fig. 13, i.e. deeply embedded inside multiple different functional units, the pervasive functionality is logically coded together in one hierarchy like any other functional unit. In a subsequent Morph-Hier step, it is then like the other functional units, mapped to a physical implementation. Fig. 13 shows such a physical implementation of the functional units and the pervasive logic.

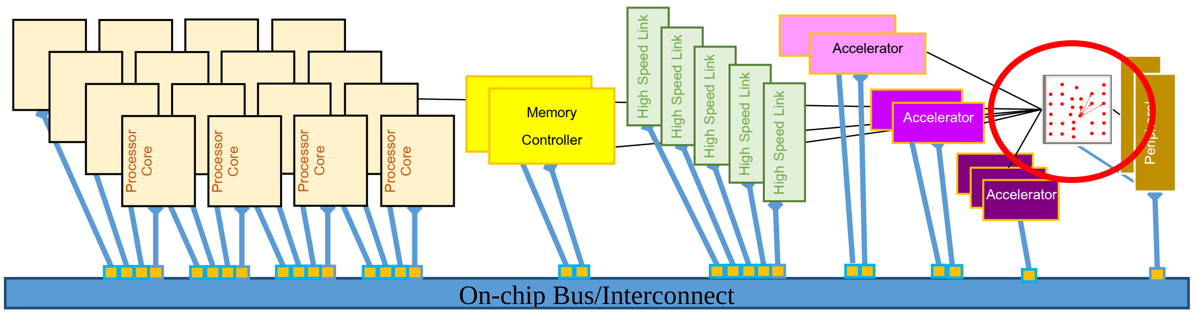


Figure12. Separation of pervasive logic (red block) from functional units, enabled by Morph-Hier.

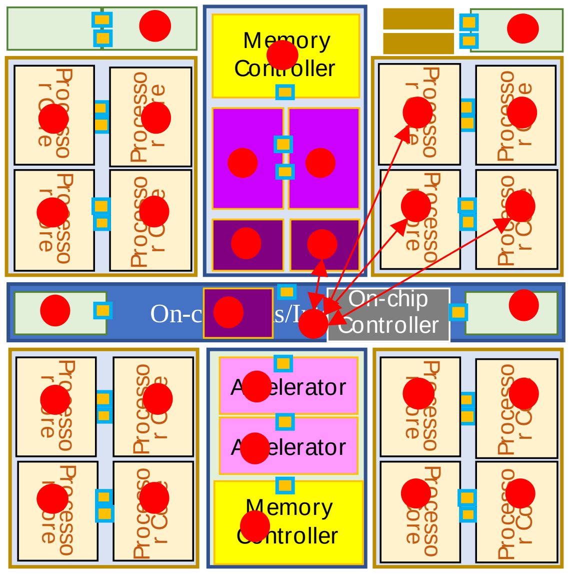


Figure13. Morphed Design Point using Morph-Hier:

- Exemplary illustration of distributed pervasive logic
- Red arrows are only shown to a few locations to simplify the picture (connections will go to all red dots)

VI. Results

Morph-Hier is built on top of the Verific Compiler [7], using its rich set of VHDL manipulation APIs and fast compilation time. Morph-Hier was run on the whole chip (Fig. 10) with more than ten million lines of VHDL, executing thousands of move, clone and feedthroughs statements. The total run time is around one and a half hour, half of which is spent in compilation. In order to verify that all transformations are correct, we created a customized formal verification tool that targets all the transformations done by Morph-Hier and can run at the chip level. The total run time of the formal tool is around one hour running on the whole chip with the majority of run time spent in the compilation step.

With respect to the three major targeted areas:

1) In the part of the design where automatic grouping of small logic blocks together to create a large physical block for synthesis is used, the design team saw significant improvement in many aspects of the design. Turnaround time of large block creation for experiments reduced from a few days to a few minutes. The numbers of large physical blocks created increased tenfold from previous Power 8 chip. It resulted in measurable area saving and noticeable timing improvement. The aggregate block size reduction of a large physical block was in the range of ten to thirty percent.
2) Dynamic creation of IBM physical chiplets while preserving the original logical HDL hierarchy significantly increased the design team's productivity. Multiple chiplets were created all at once running Morph-Hier at the chip level by a person. It did not only reduce delivery time but also head count. The biggest beneficiary was the verification team as it simplified the simulation model which led to lower bug escape rate.
3) Designing the pervasive logic in a separate unit and then automatically distributing it throughout the chip leads to 50% reduction of verification bugs on the pervasive logic in comparison to the previous POWER 8 chip. This is despite the fact that the pervasive logic got more complicated.

VII. Conclusion

In this paper, we present Morph-Hier as an automatic logical to physical hierarchy mapping tool. Morph-Hier offers a variety of operations that include instance move, wrapper creation, port connection optimization, and feedthroughs creations. This makes it of great value to applications that require mapping to physical hierarchy and exploration of different component placement and connection scenarios. Physical design engineers can now apply, independently from logic and verification engineers, the desired hierarchical design changes by writing their own recipe files. We illustrate the use of Morph-Hier on Large Synthesizable Block creation, chiplets creation, and pervasive application in IBM Power 9 chip. The tool introduced major improvements in designer productivity, bug reduction, and verification time. Moreover, the tool ensures correctness by running equivalency checking algorithms on the input and output designs.

References

[1] Synopsys., Spyglass, https://www.synopsys.com

[2] IBM Corp., https://www.ibm.com/systems/power/openpower

[3] M. Meier, et al., "AspectVHDL Stage 1: The Prototype of an Aspect-Oriented Hardware Description Language", in *Proceedings of the 2012 workshop on Modularity in Systems Software (MISS'12)*, March 2012.

[4] M. Safieddine, et al., "Methodology for Separation of Design Concerns Using Conservative RTL Flipflop Inference", *Design and Verification Conference and Exhibition*, March 2015.

[5] D. Fahland, et al., "Declarative versus Imperative Process Modeling Languages: The Issue of Understandability", T. Halpin et al. (Eds.): BPMDS 2009 and EMMSAD 2009, LNBIP 29, pp. 353–366, 2009. Springer-Verlag Berlin Heidelberg 2009.

[6] S. Pllana, and F. Xhafa, Programming Multicore and Many-core Computing Systems. John Wiley & Sons, 2017.

[7] Verific Design Automation, http://www.verific.com