GRIP: Grammar-based IP Integration and Packaging for Acceleration-rich SoC Designs

Munish Jassi
Institute for Electronic Design Automation
Technical University of Munich
munish.jassi@tum.de

Daniel Müller-Gritschneder
Institute for Electronic Design Automation
Technical University of Munich
daniel.mueller@tum.de

Ulf Schlichtmann
Institute for Electronic Design Automation
Technical University of Munich
ulf.schlichtmann@tum.de

ABSTRACT

Increased hardware IP reuse is required to meet the productivity demands for future complex Systems-on-Chip (SoCs). Nowadays, IP integration is enabled using standardized meta-data formats such as IP-Xact. We present a new concept called Grammar-based IP Integration and Packaging (GRIP), which additionally encodes design integration knowledge into a set of graph re-writing rules using standard IP-Xact. These GRIP rules are packaged into a domain-specific library of IP blocks. The library can be supplied by an IP provider along to an SoC architect. An integration tool can automatically use the GRIP rules to search the design space using the integration knowledge of the IP provider. The tool generates all design alternatives with different trade-offs for the SoC architect. We demonstrate the GRIP approach on a computer vision IP library for FPGA-based SoCs. Eighteen functional design alternatives are automatically generated within a few hours (mostly for Synthesis) requiring no hardware IP integration knowledge.

1. INTRODUCTION

IP Reuse is the major driver for increased productivity in System-on-Chip (SoC) design. Nowadays, IP integration is either done manually based on natural language documentation or with the support of meta-models such as IP-Xact [7] or Pcore [4]. These meta-information formats provide additional knowledge for IP integration, e.g. about the interfaces of IP blocks, in a machine-readable format such as XML. According to the ITRS 2011 [8], Intellectual Property (IP) reuse is estimated to reach 90% by 2020 compared to 54% reuse in today's SoC designs. As we target these high reuse values in coming years and move towards accelerator-rich SoCs, we expect the major work of SoC architects will be the integration of IP blocks provided in domain-specific IP libraries. Yet, even with this strong trend towards IP reuse, there is a clear lack of research, which looks beyond current IP integration methodology.

There are several challenges involved in the efficient integration of HW blocks provided by another party, the IP provider. For domain-specific SW libraries such as OpenCV for computer vision, it is relatively easy for the provider to clearly define interfaces for each function using programming-language-specific parameter declarations. In contrast, an optimal integration using a larger domain-specific HW IP library is much more challenging. The IP blocks may be arranged in many different ways in acceleration-rich SoCs. Sub-optimal integration may lead to scenarios, in which the gain of HW acceleration is limited due to the associated communication overheads. Additionally, integration of HW IP involves many details. One may need to define correct bus-interconnections, control, I/O and synchronization signals as well as configuration parameters. This can effectively translate into hundreds of wire signals and parameters which are required to be taken care of at Register Transfer Level (RTL). Even with model-based approaches, the challenge for the SoC architect is to acquire adequate knowledge from the IP provider to correctly and optimally integrate new IP for his target application. Another challenge is to adapt the application SW to accommodate for HW changes. Depending on the designer's expertise building a system with a new IP and investigating the inherent design trade-offs may take up to weeks.

In this work, a novel concept for IP packaging and integration called "Grammar-based IP Integration and Packaging" (GRIP) is explored to accelerate this process to hours. It consists of two major contributions: Firstly, on top of traditional approach of packaging HW IP component meta-information, we are proposing to encode additional meta- information for a complete design space exploration of a domain-specific HW IP library. This knowledge is encoded by a set of rules. The key point is, that these rules are created by the IP provider, who understands the requirements and trade-offs of his IP library. The rules allow to re-write graphs based on the theory of generative graph grammars. Each graph-rewriting rule describes exactly one architectural change of the SoC. All the rules can be defined using the IP-Xact standard such that no extension of existing IP packaging formats is required. GRIP IP packaging has two major advantages: First, it provides a way to the IP provider to formalize and encode his knowledge about optimal IP integration. Second, it provides a way for an error free knowledge exchange to the SoC architect. As a second contribution, we present a tool based on the Eclipse Modelling Framework (EMF), which is capable of loading the rules provided in the new GRIP Package. The IP-Xact descriptions of the rules are transformed into an internal graph representation. Additionally, the initial SoC architecture is transformed into a graph to allow the use of graph grammar theory. The iterative application of the rules in different combinations leads to complex architectural changes, and an automatic architectural Design Space Exploration (DSE). Additionally, a SW library is provided to the SoC architects to interface the HW from the target application. The tool has some major advantages: Requiring no HW knowledge, the SoC architect is supplied with all possible architectures, which were implicitly - by rule definition - identified by the IP provider to be useful. The progressive application of rules mimics designer's work flow and makes it easy for the

WOODSTOCK '97 El Paso, Texas USA

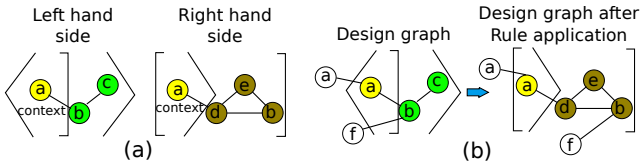

Figure 1: a) Graph Grammar rule b) Application

SoC architect to back trace any change, which was automatically applied. The SoC architect can, based on his or her own expertise, interact with the DSE process at any time, e.g., by integrating own IP at a certain stage. Additionally, EMF provides code generation facilities, which are used to quickly obtain an FPGA-based SoC implementation.

The remainder of the paper is structured as follows. In Sec. 2, we discuss related work. Background on the used graph grammar theory is given in Sec. 3. The new IP packaging approach is shown in Sec. 4. In Sec. 5, the demonstrator integration tool based on EMF and the integration process are presented. Finally, in Sec. 6, the tool is demonstrated on a vision acceleration library for FPGA-based SoCs using the Zynq chipset. We are able to run the complete DSE process and obtain 18 architectures within hours. Sec. 7 concludes.

2. RELATED WORK

IP-Xact is an IEEE standardized (IEEE 1685-2009 [7]) XML schema for IP integration. It is provided by the Accellera Initiative [1]. The benefits of IP-Xact verification and quality assurance for industrial applications are shown in [20], it describes the framework for parametrization and integration of IP cores using IP-Xact. Work from [25] presents the methodology to generate dynamic partial reconfigurable system from IP-XACT. These works outline the benefits of using IP-Xact, e.g., the meta-information on component interfaces to check legality of connections. Our approach benefits from the same advantages as we are also IP-Xact-based.

Nowadays, model driven approaches are seen as the most advanced methods for IP integration. An overview of such works is presented in [26]. Two tools, Metropolis and Ptolemy are described, which focus on model-driven and platform-based design. Yet, these approaches still require the SoC architect to understand the inherent trade-off and requirements of third-party IP blocks. Thus, these methods are very valuable for system design and early prediction but do not solve the discussed challenges of future IP reuse. Another approach in this direction was presented in [22], in which High-level synthesis is used to obtain HW accelerator blocks. This is also a top-down flow, which cannot make use of third-party IP blocks. Another direction is to run optimization heuristics that search the design space for the optimal SoC architectures. There are a wide range of heuristics available to optimize different design goals for SoCs. In the work of [21] and [9] genetic approaches are used. The work in [16] uses the Ant Colony Optimization approach. Particle Swarm Optimization is used in [11]. In [10], the exploration is described as a Markov Decision Process (MDP). In the work [23] a binary description of the graph is used for optimization. In [17], the tool Platune is presented, that optimizes parameters of a SoC platform based on cluster-based exploration method. Yet, these methods have no knowledge about the design intent behind an IP HW library. The IP provider may already have a good understanding of the optimal architectures, which need to be re-discovered by the optimization heuristics. Also, usually, the optimization process is not very transparent to the architect.

In the following sections, we show how the GRIP approach improves this situation for future IP reuse-based SoC design.

3. BACKGROUND

A generative graph grammar [13] describes an iterative process of graph re-writing steps, in which sub-graphs are continuously replaced by new sub-graphs. Graph grammars have already been used in computational mechanics for synthesizing mechanical systems [18] [19] or in software configuration [12]. In the work from [6][2], tools for graph-grammar-based design are presented. To the best of our knowledge, our work is the first one that uses graph grammar theory in the domains of IP Integration and accelerator-rich SoC design.

The graph grammar describes re-writing steps on an existing graph, the so-called *design graph*. Each graph-re-writing step is driven by the application of a pre-defined rule. The rule consists of a left-hand-side (LHS) pattern graph and a right-hand-side (RHS) pattern graph. The common elements between the LHS and RHS pattern graph are called the context. The context can be used to describe conditions, under which the rule can be applied. Only when the complete LHS pattern graph - including the context - is found in the current design graph, then the rule can be applied in a three stage process [14]: *Recognize:* In this step, the design graph is searched for the LHS pattern graph of all defined rules. *Choose:* From all rules, for which a LHS pattern graph match was found in the recognition step, one rule is chosen. *Apply:* The matching LHS pattern graph of the chosen rule is replaced with the RHS pattern graph in the design graph keeping the context unchanged.

Fig. 1 illustrates the application of the rule. In Fig. 1(a), a simple rule is defined. Fig. 1(b) shows how the application of the rule on an example design graph leads to a re-written design graph with new structure. As can be seen, additional information is part of the rule, which is not shown in the figure. For example, node d inherits the connection to node f from node b. These definitions are encoded by adding node attributes. These attributes are very useful in IP integration, as they allow to define how additional I/O, synchronization and control signals should be re-connected after an architectural change. How this is done is illustrated in the next sections.

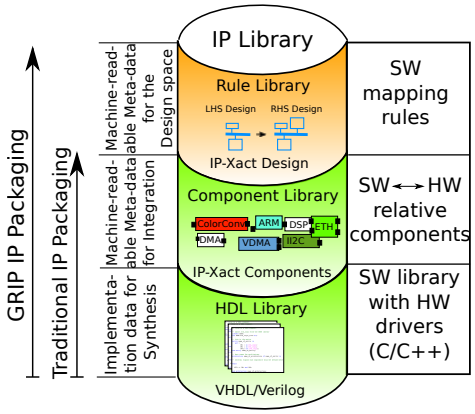

Figure 2: GRIP IP packaging incorporating machine readable design space meta-data

4. GRIP IP PACKAGING

Fig. 2 shows the GRIP IP packaging concept. The core of any HW IP library consists of the RTL implementations of HW components and their HW driver functions. On top, IP-Xact provides the meta-model for abstracting the interface and configuration information about the components. In addition to this traditional packaging, GRIP rules add meta-data information about the design space to the IP library package as is illustrated in Fig. 2.

The additional information is encoded into two IP-Xact *design* objects. An IP-Xact design object describes interconnections between and configurations of IP-Xact components. Bus ports use pre-defined logical and physical signal definitions. The signal connections among modules are represented by adHocConnections,

which have directional and connectivity properties similar to that for bus ports. These signals include synchronization, I/O, control or other signal connections. Further, each component has parametric properties, which are the IP parameters key-value pairs.

In the GRIP approach, one such IP-Xact design represents the LHS pattern, the other the RHS pattern. Even though it may appear at first glance complex to encode the design space with such rules, actually it is quite straight-forward. In a manual design, one would also not right away integrate several IP blocks at once into an SoC. Instead, the IP blocks are added sequentially, the system is reconfigured, and new architecture is synthesized, evaluated and tested. The rules formalize this flow and provide it to the SoC architect in a machine-readable format.

In the LHS patterns, the IP provider must formalize the required IP blocks and their configuration, which form the prerequisites for a certain integration step. In the RHS pattern, the IP provider must encode the correct integration changes, such as newly added bus interfaces with connected IP blocks, changed block configurations and the new connection of I/O synchronization and control signals. We illustrate the process with three examples in Fig. 3, which show the graphical representation of the IP-Xact design files. Model-driven IP-Xact tools such as Kactus [3] can be used to generate such graphical representations of IP-Xact files.

It can be seen from the LHS, that the rules target the integration of HW acceleration into a vision-display system based on an ARM SoC chipset. The context, included in both sides, includes a camera interface (camIF), display output (hdmi), a software processor system(ps7_system) and AXI buses (axiX). The first two are included because adding HW acceleration only makes sense after the vision line from camera to display is set up. The later assures that the system is based on an ARM-based SoC system, that the IPs can be configured by the HW drivers running on the processor via the buses and that the IPs can access the main memory. The buses axi0, axi2, axi3 has AXI protocol, axi1 (red) has AXILITE protocol and is used for configuration.

The LHS side shows how an IP provider can encode his design intent. Rule 1 shows how HW acceleration (ER_top) with a private Video Direct Memory Access Module (VDMA1) is added on a dedicated bus. Rule two uses an existing bus. Rule 3 pipelines a newly added HW accelerator with an existing one and reconfigures the VDMA blocks such that one reads the inputs (VDMA1) while the other writes the outputs (VDMA2). The connection is done via axis (green), which has AXISTREAM protocol. As can be seen, e.g, there is no rule that would add an independent bus or accelerator without VDMA, as this is not seen as useful by the IP provider. E.g., VDMA is required as the axi2 and axi3 are not mapped on the processor memory. Alone with these three rules and a set of HW accelerators, a whole range of possible architectures can be generated for different vision-display applications. Yet, the actual encoded rules require some more effort. Fig.3 also shows the number of parameters, which need to be re-configured for each block in each integration step, and additionally, the number of additional non-bus signals which require new connections. The GRIP packaging is not restricted to computer vision (CV) domain, but we use CV to show the potential of our approach for SoC designs.

5. GRIP INTEGRATION AND DSE TOOL

In this section, we discuss a tool which can use the GRIP packaging information. First, we discuss the rule application process, then the DSE process, and finally the EMF-based demonstrator implementation.

5.1 Execution of Rules

Each rule execution starts from a *candidate SoC architecture* given as an IP-Xact design. Internally, a model transformation is applied to transform the IP-Xact design into an architectural graph

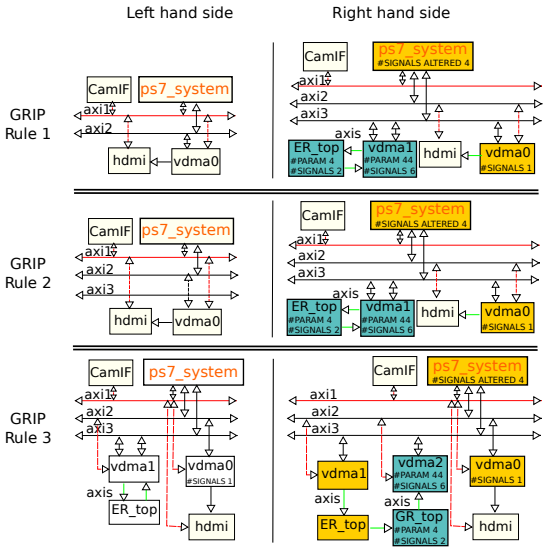

Figure 3: Example GRIP rules to add HW accelerator a) on dedicated Bus; b) on existing Bus; c) in pipeline with other HW accelerator

representation (AGR). The AGR is a port-based graph representation. In the AGR, the nodes represent IP blocks and the edges represent port bindings. Both the communication components (e.g. buses) as well as the computational components are represented by the nodes. The ports are labelled with the information provided by IP-Xact, e.g. bus ports are categorized into master port, slave port, mirrored-master port, or mirrored-slave port. This port information and the block configuration parameters are represented as node and port labels in the graph. Labels in our case are triples *<datatype, name, value>*, which describe the properties of each node. Thus, the AGR carries the same information as IP-Xact design objects and the IP-Xact file can be generated completely from the AGR and vice versa. Applying the rules directly on the IP-Xact representation is not favourable as the data structure is not suitable for matching and replacement algorithms. In contrast, there exist already mature and very efficient graph re-writing algorithms from graph grammar theory [15]. These can be directly applied to the AGR. For this, also the LHS and RHS of the rules are transformed to their AGRs. The execution of rules on a candidate AGR is in our case a 5-step process:

Recognize: Graph matching algorithms are used to find all sub-graphs in the current design AGR, that match the LHS AGR of any defined rule. As was described in the previous section, context is used to describe the conditions under which any rule can be applied. Thus if the LHS is found, the rule application should result in a useful modification of the SoC architecture.

Choose: Often more than one rule is applicable for the current candidate. Either one applicable rule can be chosen by the tool for an automated DSE or the SoC architect can interact and select rules.

Apply: The chosen rule is applied. The nodes of the sub-graph of the LHS are replaced by nodes of the RHS pattern. Configuration parameters are changed by adapting node labels. Buses and signals are re-connected by removing and adding edges to the AGR and the new AGR is generated.

Check: Next to the rules, the IP-Xact component descriptions hold the data on the legality of IP integration. The newly generated AGR is transformed back to IP-Xact and checked against the IP-Xact component meta-data. This is done because the IP provider might not foresee that some rule might lead to an illegal design, e.g., the maximal number of ports on a bus component is exceeded.

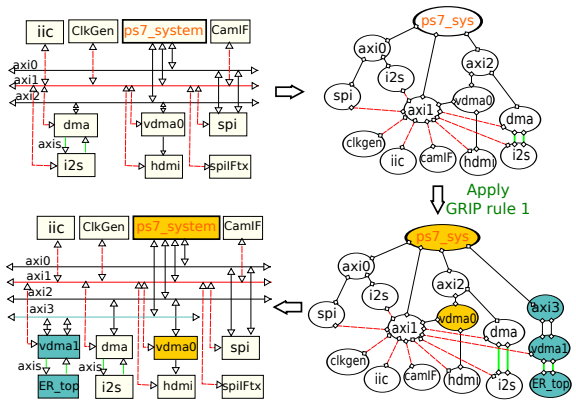

Figure 4: Model-to-model transformation from IP-Xact design to AGR and GRIP rule application for HW accelerator integration

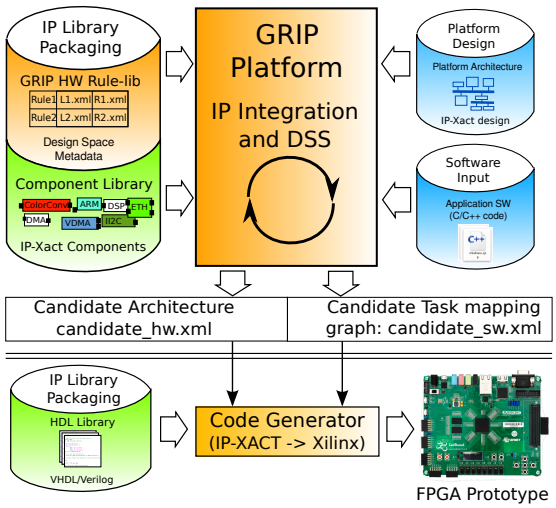

Figure 5: GRIP demonstrates integration tool

Also, the later described DSE process allows interaction with the SoC architect. In this step, it can also be detected if manual intervention of the SoC architect leads to an illegal design.

Evaluation: If the newly generated candidate IP design file is legal, the implementation files, e.g. for FPGA synthesis are generated. The design is synthesized and executed. A performance report is written and read back into the tool to report the current design trade-off to the SoC architect.

Fig. 4 illustrates the rule application. Starting from the candidate IP-Xact design, the AGR is generated and rule 1 from Fig. 3 is applied to add a HW accelerator. The new IP-Xact design is generated from the modified AGR.

5.2 DSE Process

The execution of the rules is done iteratively by repeating steps one to five. This iterative execution of rules progressively generates new architectures by integrating the available IP blocks and reconfiguring the connectivity, leading to a DSE over the provided IP library. The root node of the generated search tree is the initial architecture candidate provided by the SoC architect. Each edge in the tree represents the application of a rule as shown in the previous section. Branching occurs if more than one rule is applicable. To explore the tree, breadth-first can be used to first generate solutions near the initial candidate and depth-first to generate first solutions far away from the initial solution. The exact search order and search depth is chosen by the designer. If it is not restricted, the process will result in a complete DSE.

The search tree gives the SoC architect the ability to transparently backtrack any change resulting from automatic rule application. Additionally, the SoC architect can interact in any possible way with the exploration, e.g., by manually integrating own IP in a certain candidate architecture and restarting the exploration. If he provides IP-Xact component data for his own IP, legality and cross-compatibility are automatically checked. The same is true if two third party IP libraries are used in combination.

5.3 Implementation

The GRIP platform is implemented based on the Eclipse Modelling Framework (EMF). We used the XML Schema for IP-Xact to automatically generate the EMF ecore model and utility functions required for working with IP-Xact.

The GRIP tool broadly consists of five parts:

The *IP library parser:* This reads the IP-Xact IP library components and associated GRIP rules.

Model2model Transformation: The input design, IP library, and grammar rules are internally transformed into the AGR model and back.

Rule application: This engine finds matches for GRIP rules and generates new AGRs. The checks against IP Xact component data are also done to assure that the new generated designs are legal. For the graph pattern matching, the so-called Centralized Algorithm is applied [24]. For the rule application, the custom algorithm in Sec. 5.4 is used.

Code generation: This translates an IP-Xact design into implementation files for FPGA prototyping. At the moment, we support code generation from IP-Xact for Xilinx FPGA-based SoC synthesis.

SW adoption: This changes the application SW and includes the HW drivers for the newly added HW accelerators, e.g., configuration and control functions for IP blocks. At the moment, we support only bare-metal implementations.

Fig. 5 shows a complete integration tool flow for the GRIP approach. The IP-supplier must provide the GRIP packaging. The SoC architect starts with an initial architecture defined in IP-Xact, e.g., the platform architecture of the used SoC FPGA. Additionally, he must supply the application source code in order to define the target application. For this, the IP provider also provides a SW library that is used by the SoC architect. This allows to identify, where in the target application functions are used that can be accelerated by the IP blocks provided in the domain-specific IP library. Additionally, if also SW implementations for designs are provided, then the DSE process can also explore different HW/SW mappings. It also allows to generate performance data even if not all required HW IP blocks are yet integrated, thus, implementing the complete application from the beginning. Else it might be that several rules might be supplied first, before all IP blocks are implemented in the design. The tool provides the results from the DSE to the SoC architect. For this, the designs are executed on the FPGA and data from performance monitoring IP is returned to the tool.

5.4 Graph-rewriting algorithm

The algorithm to apply the rules is shown in Algo. 1. The inputs are $D(LHS, RHS)$ and $G(V_G, E_G)$. LHS and RHS are the LHS and RHS pattern graphs of the AGR of the rule D, which is applied. The graph $G(V_G, E_G)$ is the AGR of the current candidate that is explored. In contrast to existing graph re-writing algorithms we have to take special care about the labels, as we need to re-configure blocks and construct a correct wiring of bus, control and synchronization signals. This is important, otherwise feasible solutions become non-functional because the signals are not connected correctly.

This is done as follows: The set l contains the labels for all the

Algorithm 1: Application of a rule

Input : Rule $D : LHS(V_L, E_L), RHS(V_R, E_R)$ and model graph $G(V_G, E_G)$
Output : New model graph $Q(V_Q, E_Q)$

if *match*$(L, G) == LHS$ **then**
  foreach *node* $u \in V_L \backslash V_{bus}$ **do**
    $sim(u) := \{w | w \in V_G, l_G(w) = l_L(u)\}$;
    $V_G := V_G \backslash \{w\}$;
  foreach $v \in V_R$ **do**
    $V_G := V_G \cup \{v\}$;
    $E_G := E_G \cup$
      $\{(\text{sim}(v), w) : v \in V_R, w \in \text{sim}(\text{post}_R(v))\}$;
  $Q := G$;
return Q

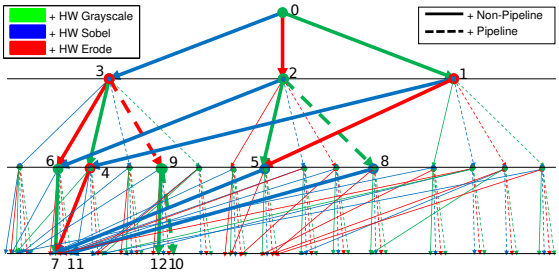

Figure 6: Design search tree for the case study.

attributes for the node. The set $sim(u) \in G$, contains all the candidate nodes, which have identical labels as one node $u \in LHS$. In the matching phase, the start pattern LHS was already matched to G using the Centralized Algorithm. If the complete match is found, then for all $u \in V_L$, $sim(u)$ are removed from G except V_{bus}, *bus node*, and all $v \in V_R$ are added to G. New edges are created for the newly added nodes in accordance to RHS graph, to generate the new model graph Q. While creating the next edge, the algorithm reads the connected ports of the concerned node and refer back to library instance of that node to find available unconnected port. On the returned unconnected node the new edge is drawn, which makes the new connections correct by construction. The remaining interconnections for buses and signals are kept unchanged.

6. CASE STUDY

The GRIP concept is demonstrated on a vision-display application, which captures video data and sequentially performs three vision steps on the input frames: sobel filtering (SO), erosion filtering (ER), and grayscale conversion (GR). The target platform is the ZedBoard [5] with Xilinx Zynq chipset. It has a hard dual-core ARM cortex_a9 processor, 512MB DDR, MCU, multiple AXI4 and AXILite buses, and programmable FPGA fabric for HW acceleration. Vision tasks load even such powerful embedded processors heavily, therefore this domain is highly suitable for HW acceleration.

The HW IP blocks and SW functions for these three vision tasks (SO, ER, GR) were implemented in an IP library. The library references additional Xilinx IP cores used for system implementation such as VDMAs. This is also a nice feature of the proposed approach, as the IP provider can expect that these cores are available via Xilinx.

The implemented rules include the three examples shown in Fig. 3 for integration of HW acceleration. Additional rules are provided for exploring more configurations such as enabling power optimizations provided by the FPGA synthesis tool, changing data widths of buses and IP interfaces between 32bit/64bit and enabling Accelerator Coherency Port (ACP) optimization.

Figure 6 shows the resulting DSE search tree. A limited enumeration of solutions and ordering these in a tree results in 81 architectures. Sequential application of the GRIP rules starting from the initial SoC architecture reduced the DSE search space to 12 desirable architectures (bold sub-tree). This figure should illustrate, that it makes sense that the IP provider restricts the design space by defining the rules accordingly. This avoids that many non-desirable solutions need to be investigated.

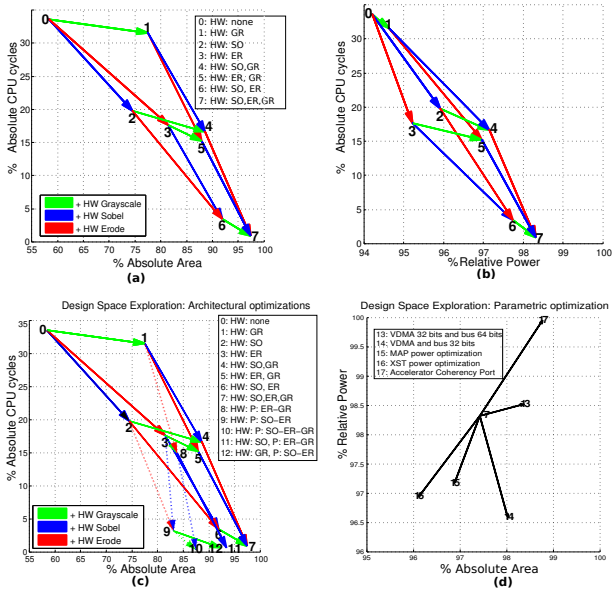

Figure 7: a) CPU load vs. area; b) CPU load vs. power, for non-pipelined configurations; c) CPU load vs. Area for architectural DSE (12 configs); d) Power vs Area for parametric DSE (5 configs)

As was pointed out, the tool supports an interactive integration and DSE process. This is illustrated with a example DSE flow. First, the SoC architect chooses to apply rule 1 and rule 2. This leads to the integration of HW accelerators without pipelining. Performances were extracted from FPGA monitor data and provided to the SoC architect. Figure 7 (a) and (b) shows the trade-off for the 7 non-pipelined architectures. Arch 0 is the initial architecture with all tasks in SW, arch 1,2,3 have one HA each, GR, SO, ER respectively, arch 4,5,6 have two HAs GR-SO, GR-ER, ER-SO respectively, arch 7 has all three tasks in HW. The Pareto visualization help to analyse the overall effect for interdependent design parameters. Arch. 2,6,7 and 3,6,7 are the Paretos in Fig 7(a) and Fig. 7(b) respectively. Higher power usage of SO accounting for larger area is overshadowed by power reduction because of reduced CPU load.

Starting from this point, the SoC architect could decide to explore pipelined architectures by applying rule 3. This leads to 5 new architectures. Arch 8, 9 are two HAs in pipeline ER>GR, SO>ER respectively, while arch 10 is all three in pipeline GR>SO>ER. In arch 11, 12 ER>GR are in pipeline and SO in parallel, SO>ER in pipeline and GR in parallel respectively. The performance values are shown in Fig. 7 (c). As can be seen, there are benefits in terms of area because less DMA resources are needed. Yet it also restricts flexibility because the software cannot access the video data in between two processing steps. The SoC architect could, thus, decide to stick to Arch 7 with three non-pipelined accelerators. On this architecture, the additional rules for power, bit width and coherency ACP optimization are applied to obtain another 5 architectures. The resulting performance values can be seen from Fig. 7 (d). As can be seen, XST power optimization brings additional area and power benefits. Fig 8 gives the overall performance report for power, area and CPU load for all so-far explored architectures. The SoC architect can stop here or run the tool further to explore the complete

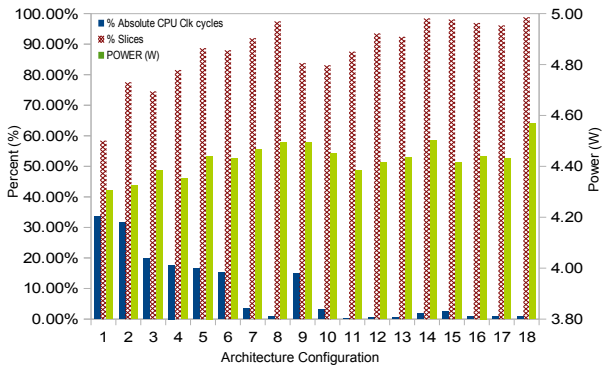

Figure 8: Power, area, and CPU load comparison for all 18 architectures.

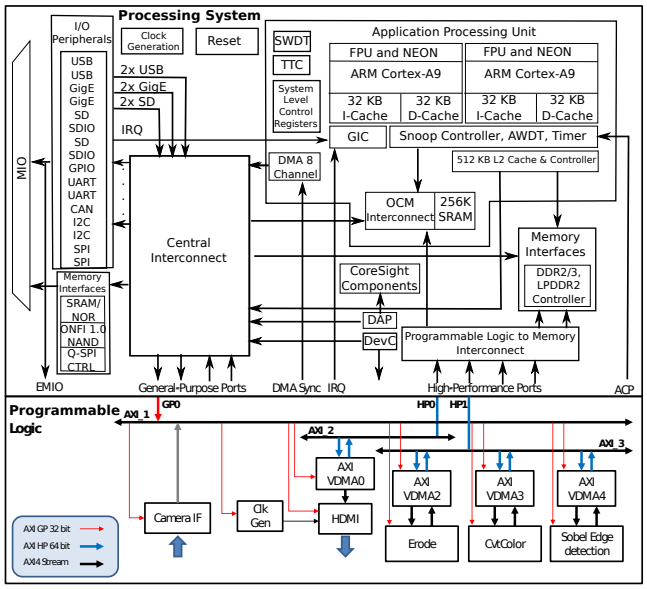

Figure 9: Example architecture on ZedBoard.

design space.

Figure 9 elaborates the FPGA prototype implementation of arch 7. The ARM processors (667MHz) fetch the video frames (640x480, 15fps) via *camera interface* module attached to AXI_1 bus (AXILite, 100MHz) and writes it to DDR. Then, it passes the control to VDMA4 on AXI_3 (AXI4, 150MHz) to fetch the captured frame for SO Hardware Accelerator (HA) for processing, followed by VDMA2 takes over the control on AXI4 bus to do erosion filtering and lastly VDMA3 to sequentially process the output from ER to do gray scale conversion. All these processed images are written at different addresses in the DDR, and HDMI module on AXI_2 (AXI4, 150MHz) reads these processed images to display it on the external display monitor.

The complete DSE process including synthesis for FPGA took 21 hours, which were mainly required for the FPGA synthesis. The process required no IP integration knowledge.

7. CONCLUSIONS

IP reuse is of utter importance for future SoC design. The presented work showed novel ways to look beyond traditional IP integration methodology. We believe even more research work in this direction is required to reach the IP reuse values targeted in the ITRS roadmap.

8. REFERENCES

[1] Accelera website. http://www.accellera.org.

[2] eMoflon. http://www.moflon.org/.

[3] Kactus2 tool. http://www.funbase.cs.tut.fi.

[4] Xilinx Pcore. www.xilinx.com/tools/coregen.htm.

[5] ZedBoard. http://www.zedboard.org.

[6] Booggie tool. http://www.booggie.org, 2009.

[7] IEEE 1685-2009 IP-Xact. http://standards.ieee.org, 2009.

[8] International Technology Roadmap for Semiconductors, www.itrs.net. 2011.

[9] G. Ascia, V. Catania, and M. Palesi. A ga-based design space exploration framework for parameterized system on-a-chip platforms. *IEEE Trans. Evolutionary Computation*, 8(4):329–346, 2004.

[10] G. Beltrame, L. Fossati, and D. Sciuto. Decision-theoretic design space exploration of multiprocessor platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, July 2010.

[11] A. Bhattacharya, A. Konar, S. Das, C. Grosan, and A. Abraham. Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm. *International Conference on Complex, Intelligent and Software Intensive Systems*, 2010.

[12] Thomas Buchmann, Bernhard Westfechtel, and Sabine Winetzhammer. The Added Value of Programmed Graph Transformations, A Case Study from Software Configuration Management. In *Applications of Graph Transformations with Industrial Relevance*, volume 7233 of *Lecture Notes in Computer Science*, pages 198–209. 2012.

[13] A. Chakrabarti, K. Shea, R. Stone, J. Cagan, M. Campbell, N. V. Hernandez, and K. L. Wood. Computer-based design synthesis research: An overview. *J. Comput. Inf. Sci. Eng.*, 11, 2011.

[14] S.C. Chase. A model for user interaction in grammar-based design systems. *Automation in Construction*, 2002.

[15] Hartmut Ehrig and Julia Padberg. Graph grammars and Petri net transformations. In *Lectures on Concurrency and Petri Nets*, pages 496–536. Springer, 2004.

[16] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *Computer-Aided Design of Integrated Circuits and Systems*, June 2010.

[17] T. Givargis and F. Vahid. Platune: A tuning framework for system-on-a-chip platforms. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 21(11), 2002.

[18] Corinna Königseder and Kristina Shea. Strategies for Topologic and Parametric Rule Application in Automated Design Synthesis using Graph Grammars. American Society of Mechanical Engineers (ASME), 2014.

[19] Corinna Königseder and Kristina Shea. Systematic rule analysis of generative design grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 28:227–238, 8 2014.

[20] W. Kruijtzer, P. van der Wolf, E. de Kock, J. Stuyt, W. Ecker, A. Mayer, S. Hustin, C. Amerijckx, S.de Paoli, and E. Vaumorin. Industrial IP Integration Flows based on IP-XACT standards. *DATE*, 2008.

[21] M. Li, S. Azarm, and V. Aute. A multi-objective genetic algorithm for robust design optimization. *Proc. of GECCO05*, pages 771–778, 2005.

[22] Hung-Yi Liu, Michele Petracca, and Luca P. Carloni. Compositional System-Level Design Exploration with Planning of High-Level Synthesis. *DATE*, March 2012.

[23] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich. Combined system synthesis and communication architecture exploration for mpsocs. *DATE*, April 2009.

[24] S. Ma, Y. Cao, J. Huai, and T. Wo. Distributed graph pattern matching. *WWW*, 2012.

[25] G. Ochoa-Ruiz, El-Bay Bourennane, Hassan Rabah, and Ouassila Labbani. High-level modelling and automatic generation of dynamicaly reconfigurable systems. *DASIP*, 2011.

[26] A. S.-Vincentelli, S. K. Shukla, J. Sztipanovits, G. Yang, and D. A. Mathaikutty. Metamodeling: An Emerging Representation Paradigm for System-Level Design. *IEEE Design & Test of Comp.*, May 2009.