# IP-XACT Extensions for Reconfigurable Computing

Razvan Nane*, Sven van Haastregt†, Todor Stefanov†, Bart Kienhuis†, Vlad Mihai Sima* and Koen Bertels*
* Computer Engineering Lab, Delft University of Technology
Email: ⟨r.nane, v.m.sima, k.l.m.bertels⟩@tudelft.nl
†LIACS, Leiden University
Email: ⟨svhaastr, stefanov, kienhuis⟩@liacs.nl

*Abstract*—**Many of today's embedded multiprocessor systems are implemented as heterogeneous systems, consisting of hardware and software components. To automate the composition and integration of multiprocessor systems, the IP-XACT standard was defined to describe hardware IP blocks and (sub)systems. However, the IP-XACT standard does not provide sufficient means to express Reconfigurable Computing (RC) specific information, such as Hardware dependent Software (HdS) meta-data, which prevents automated integration. In this paper, we propose several IP-XACT extensions such that the HdS can be generated and integrated automatically. We validate these specific extensions and demonstrate the interoperability of the approach based on an H.264 decoder application case study. For this case study we achieved an overall 30.4% application-wise speed-up and we reduced the development time of HdS from days to a few seconds.**

## I. Introduction

A widely adopted practice within Reconfigurable Computing (RC) design is to accelerate part(s) of applications, using custom hardware architectures, that are specifically tailored for a particular application. These specialized architectures can be *Intellectual Property* (*IP*) blocks written at the *Register Transfer Level* (*RTL*) by a designer, or IP blocks generated by a High Level Synthesis tool from a functional specification written in a *High Level Language* (*HLL*) [7]. To cope with the diversity of IP blocks coming from different sources, IP-XACT [6] was introduced. Using IP-XACT, hardware components can be described in a standardized way. This enables automated configuration and integration of IP blocks, aiding hardware reuse and facilitating tool interoperability [3].

In a Hardware/Software (HW/SW) system, connecting the different HW and SW components, using for instance buses or point-to-point connections, is not sufficient to fully implement a system. Typically, a SW component connected to a HW component needs a driver program, also known as *Hardware dependent Software* (*HdS*) [1], to control the HW component. IP blocks that can be controlled from a SW component are typically shipped with particular HdS to ensure proper control from SW. However, in RC systems, the IP blocks are automatically generated by HW tool-chains for application kernels selected for hardware acceleration. Therefore, the HdS driving these new hardware blocks has to be generated automatically as well. The compilation process in such RC systems, i.e., from HLL application source code to a combined HW/SW executable, is done by different tools, such as partitioning, mapping and HW/SW generation tools. This implies that there is no central place from where the HdS can be generated. That is, the compiler used to generate the IP has no knowledge about what HW primitives are used for example to communicate data in the system, which prevents it from generating a proper driver. This information is available, however, in the partitioning and mapping tool. Therefore, we adopt a layered solution in which different parts of the HdS are generated at different points in the tool-flow. Furthermore, to allow the tools involved in this HdS generation process to communicate seamlessly with each other, we need to describe the software requirements of each step in IP-XACT as well. One example of such a software requirement is the number of function input parameters. However, unlike the RTL constituents of an IP block, which can be already described using the current IP-XACT standard, there is no standardized way to describe the driver information for an IP.

In this paper, we elaborate on the expressiveness of IP-XACT for describing HdS meta-data. Furthermore, we address the automation of HdS generation in the RC field, where IPs and their HdS are generated on the fly, and therefore, are not fully predefined. The contribution of this paper can be summarized as follows:

- We combine two proven technologies used in MPSoC design, namely IP-XACT and HdS, to automatically integrate different architectural templates used in RC systems.
- We investigate and propose IP-XACT extensions to allow automatic generation of HdS in RC tool-chains.

The rest of the paper is organized as follows. Section II presents IP-XACT, other HdS solutions, and already proposed IP-XACT extensions. Section III describes a HdS generation case study and investigates the IP-XACT support for automation. Section IV elaborates on the identified shortcomings and proposes IP-XACT extensions to support software related driver descriptions. Section V respectively Section VI validates the automated integration and concludes the paper.

## II. Related Work

The IP-XACT standard (IEEE 1685-2009) [6] describes an XML schema for meta-data modelling IP blocks and (sub)systems. The meta-data is used in the development, implementation and verification of electronic systems. In this paper, we focus on *Component* schema for associating HdS to HW IP blocks and we focus on *Generator-Chain* schema to express compiler specific requirements. The current schema provides limited support for software descriptions. Namely, one can only attach software file-sets to a component and describe the function parameter's high level types. However, it does not offer means to assign semantics to the attached file-set and how it should be used during integration of a complete system. Furthermore, it lacks means to model tool chains in which complex software generators are to be integrated. In Section IV, we propose solutions to these problems.

The OpenFPGA CoreLib [9] working group focused on examining the IP-XACT Schema and proposed extensions for facilitating core reuse into HLLs. Wirthlin et al. [10] used XML to describe common IP block elements and defined their own schema using IP-XACT syntax. They proposed a lightweight version intended for Reconfigurable Computing (RC) systems, such as interface specifications and capturing HLLs data types information.

Other IP-XACT related research is focusing on extending the schema to incorporate semantic information about IP elements. Kruijtzer et al. [4], proposed adding *context labels* to provide additional meaning to IP-XACT components. They use this to assess the correctness of interconnections in the system. Strik et al. [5] studied aspects regarding IP (re)configurability to reuse these after a partial change of some parameters. They underline that IP-XACT is missing *expression evaluation* fields to support flagging illegal (sub)system composition. However, all proposed extensions discussed so far in this section consider only the HW IP block. As mentioned in Section I, for systems involving both HW and SW, one also needs to describe the HdS belonging to the HW IP to enable automated integration of a system. Therefore, we propose software related extensions for IP-XACT.

## III. Integrating Orthogonal Computation Models

To investigate the IP-XACT capabilities to model HW/SW co-design tool chains supporting HdS generation and tool interoperability, we used an H.264 decoder application implemented on an *Field Programmable Gate Array* (*FPGA*) as a case study. The goal is to integrate different tools and models such that we can automatically generate application specific MPSoC implementations of sequentially specified applications. To realize this, we use the Daedalus [8] system level synthesis tool set to implement an MPSoC from sequential C code. In particular, from the Daedalus tool set we use the PN compiler [13] to partition a sequential application into *Polyhedral Process Networks* (*PPN*) and ESPAM [11] to map the partitioned application onto an FPGA. Outside the Daedalus tool-set, we use DWARV [7] to automatically generate hardware IP blocks for performance critical parts of the sequential C code. We first describe the problems observed when integrating the two tools in Section III-A. Subsequently, we present our extended framework in Section III-B.

### A. IP Core Integration

In [12], the PICO compiler from Synfora Inc. [14] was incorporated in the Daedalus tool-flow. This approach was used to achieve higher performance for PPN implementations by replacing computation intensive nodes with functionally

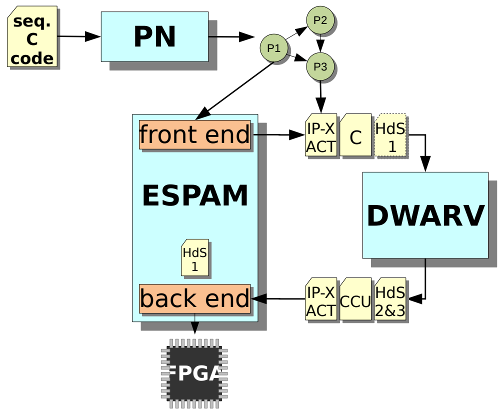

Fig. 1: H.264 Generation Tool-Chain Flow.

equivalent hardware IP cores generated by PICO from the available C code. The replacement was done smoothly, as both tools were operating under the same memory model, i.e., a distributed memory model. However, several restrictions were imposed on the C code which can be processed by the PICO compiler. For instance, each loop body could contain only one other loop. Therefore, using PICO as the hardware compiler was not feasible for the H.264 application where multiple nested loops are present. DWARV is a compiler which has less restrictions than PICO, making it suitable for generating hardware blocks for our case study.

However, integration of a *Custom Compute Unit* (*CCU*), generated by DWARV, in a PPN created by Daedalus is not straightforward. The CCU has an interface suitable for interacting with Molen [2] based platforms, which employ a shared memory model. The PPN node, on the other hand, into which the CCU has to be integrated, has multiple input and output FIFO channels typical for the stream-based distributed memory model. The challenge therefore is to find a way to specify the requirements for Daedalus such that DWARV can automatically generate the correct interface.

### B. Framework Solution

We show our solution in Fig. 1. We use the PN compiler to create a PPN from the sequential C code of the H.264 top level function. Subsequently, we use ESPAM to implement the H.264 PPN as a system of point-to-point connected MicroBlaze processors on an FPGA, as shown in the left part of Fig. 1. This means the functional part of each process is implemented as a software program running on a MicroBlaze. Based on profile information we have decided to accelerate the Inverse Discrete Cosine Transform (*IDCT*) process using a specialized hardware component. We use the DWARV C-to-VHDL compiler to generate a CCU from the C code of the IDCT function, which requires the C function to be communicated from ESPAM to DWARV.

To solve the interface mismatch problem between DWARV-generated CCUs and Daedalus' PPNs, DWARV generates a wrapper for the CCU. This wrapper provides memory to the CCU which stores the input/output channel data before/after the CCU is started/stopped.

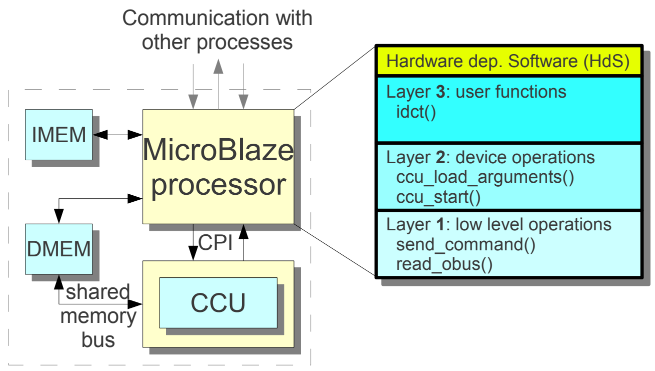

Fig. 2: Connection between CCU and processor (left) and HdS layers (right). IMEM is the instruction memory of the processor, while DMEM is the data memory that is shared between both the processor and the CCU.

The HdS controlling the CCU is structured into three different layers. The right side of Fig. 2 shows the HdS hierarchy. We distinguish platform primitives (layer 1), IP- and OS-specific driver code (layer 2) and an application layer (layer 3). The primitives in layer 1 strongly depend on the processor the HdS is running on, and the way the CCU is connected to the processor running the HdS. For instance, for one processor these primitives use memory-mapped I/O, whereas for another processor dedicated instructions are available. This information is known only by ESPAM. Therefore, the HdS layer 1 primitives are generated by ESPAM. HdS layer 2 provides functions that control the CCU by sending and receiving commands and data to and from the CCU using the primitives provided by layer 1. The separation of HdS layers 1 and 2 makes the HdS layer 2 code independent of the actual platform. HdS layer 3 provides user level functions, which are invoked by a user application to perform the task for which the CCU was designed. The functions in layer 3 only use functions provided by HdS layer 2. The HdS layer 3 function(s) provide a transparent interface to the CCU, essentially making the CCU available as a regular software function.

## IV. IP-XACT EXTENSIONS

In this section, we elaborate on the expressiveness of the current IP-XACT standard to describe the scenario presented in the previous section. Based on this analysis, we describe three possible extensions, namely hardware compiler input, HdS and tool-flow integration related extensions. We implemented the proposed extensions using the *vendorExtensions* extension construct that is already part of IP-XACT. This allows vendor specific information to be added to IP-XACT descriptions.

### A. Hardware Compiler Input

The DWARV compiler accepts a C function as input and generates a VHDL component that implements the original C function. In our case study, we send the C function together with an IP-XACT metadata description to DWARV. DWARV assumes that a function argument can be both input and output at the same time, if the argument is a pointer into shared memory. However, arguments to functions inside a PPN process are always unidirectional. For each function argument, unidirectional First In, First Out (FIFO) channels are created according to the process network topology. Therefore, we need to inform DWARV about the direction of each function argument, such that the appropriate FIFO input or output connection can be generated. We therefore add a **direction** field to the IP-XACT description that is passed along with the C file defining the function implementation. The values this field can take are: *in*, *out* and *inout*.

```
<spirit:function>
  <spirit:entryPoint>send_data</spirit:entryPoint>
  <spirit:fileRef>f-hds1_h</spirit:fileRef>
  <spirit:returnType>void</spirit:returnType>
  <spirit:argument spirit:dataType="int">
    <spirit:name>data</spirit:name>
    <spirit:value>0</spirit:value>
  </spirit:argument>
  <spirit:vendorExtensions>
    <spirit:hdstype>write</spirit:hdstype>
  </spirit:vendorExtensions>
</spirit:function>
```

Fig. 3: HdS IP-XACT extensions for layer 1.

### B. Hardware Dependent Software

Using HdS in heterogeneous MPSoCs abstracts hardware and OS details away from the application level. In our case study, we have partitioned the HdS into three different layers, as described in Section III-B. HdS layer 1 is generated by the Daedalus environment and then passed to DWARV. This enables DWARV to generate HdS layers 2 and 3 that make use of the primitives provided by HdS layer 1.

To create a semantic link between two different HdS layers, we need to specify the purpose of the functions found in HdS1. For HdS layer 1, we classify a function as *read*, *write* or *command*. An example of such a description in IP-XACT is shown in Fig. 3. The *read* identifier classifies the function as one that reads data from the *CCU-Processor Interface* (*CPI*), which has been implemented using two small FIFO buffers. The *write* identifier classifies the function as one that writes application data to the CPI and the *command* identifier classifies a function as one that writes control data to the CPI. Because hardware primitives are typically limited in number, we define a new IP-XACT type **HdS type** to establish a semantic link between layers 1 and 2.

Similarly, we can create a link between layers 2 and 3. However, layer 2 is concerned with abstracting OS specific implementation details for the custom IP block, and since there is no OS present in our case study, we leave the definition of this type as future work. Nevertheless, we imagine that this type could include identifiers for the *POSIX* standard such as opening and closing file handles.

### C. Tool Chains

To fully automate the tool flow shown in Fig. 1, IP-XACT provides means to model generator chains. For example, the current IP-XACT standard provides a *generatorExe* field which contains the executable to be invoked for a *generator*. However, we observe that IP-XACT currently lacks a way to describe the tool-specific configuration files. For example, DWARV uses an external Floating Point (FP) library description file listing the available FP cores, such that floating point arithmetic in the C code can be implemented using available FP cores. To allow seamless cooperation of different tools from different vendors, we observe the need to include tool-specific descriptions and files in IP-XACT generatorChain schema.

## V. Experimental Results

In this section, we report on two kinds of results. First, we show the applicability and usefulness in a real world application and second, we report the overall productivity gain. We base this on our experience with the H.264 case study, for which the first implementation was done manually.

### A. Validation of Approach

In our experiments, we target a Xilinx Virtex-5 XC5VLX110T-2 FPGA and use Xilinx EDK 9.2 for low level synthesis. We use *QCIF* (176x144 pixels) video resolution and a 100 MHz clock frequency. To validate the approach, we implement the H.264 decoder application twice. The first time we map all processes of the PPN onto MicroBlaze processors, which means all PPN processes are implemented as software. This serves as our reference implementation. The second time we replace the software IDCT node with a hardware version obtained using the methodology described in the previous sections. We obtain a speed-up of approximately 30.4%.

### B. Productivity Gain

Besides proving the usefulness of the approach to obtain a faster implementation of a PPN, we discuss the productivity gain observed when adopting an automated IP-XACT based approach. If the automated support was not available, manually patching the tools would have been time consuming and error-prone. Depending on the application knowledge of the system designer and application complexity, activities like writing the HdS or the CCU wrapper can take from a few hours up to even weeks. Moreover, validation may take a similar amount of time. For example, a memory map has to be specified as C code in the HdS and as VHDL in the RTL. For correct operation of the system, these two representations need to be fully consistent, which may be an important source of errors when manual action is involved. We eliminate such errors by taking the information from a central source (e.g., an IP-XACT description) and then automatically generate the different representations. This substantially reduces the time needed for validation. To fully understand the specific challenges and properly design the modifications required by the tools to enable automated integration, our first implementation of the system was manual. Based on this experience, we estimate that building a fully working system for the H.264 decoder application by hand would take one week. Using the approach described in this work, one could obtain a working system in less than an hour, which is a considerable gain in productivity.

## VI. Conclusion

In this paper, we have presented a new approach for automated generation of RTL implementations from sequential programs written in the C language. This is achieved by combining the Daedalus framework with the DWARV C-to-VHDL compiler with the aid of the IP-XACT standard. With these concepts, even different architectural templates can be reconciled. We investigated the capabilities of the IP-XACT standard to model automated integration of MPSoCs consisting of both hardware and software components. We found that the Hardware Dependent Software needed to control the hardware component cannot be described in the current IP-XACT standard. We identified three possible concepts that could be added as extensions to the IP-XACT standard to realize automated integration of HW/SW systems. Using an H.264 video decoder application we verified our approach.

## Acknowledgements

This research has been funded by the projects SoftSoC PNEI082007, Smecy 100230, iFEST 100203 and REFLECT 248976.

## References

[1] S. Yoo, M. Youssef, A. Bouchhima and A. Jerraya. *Multi-Processor SoC Design Methodology Using a Concept of Two-Layer Hardware-Dependent Software*. In Design Automation and Test in Europe (DATE'04).

[2] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov and E. Panainte. *The molen polymorphic processor*. In IEEE Transactions on Computers (November 2004).

[3] J. Wilson et al. *Industrially Proving the SPIRIT Consortium Specifications for Design Chain Integration*. In Proc. of Design, automation and test in Europe (DATE '06). pages: 142 - 147.

[4] W. Kruijtzer, E. de Kock, J. Stuyt, W. Ecker and E. Vaumorin. *Industrial IP Integration Flows based on IP-XACT Standards*. In Proc. of Design, automation and test in Europe (DATE '08). pages: 32 - 37.

[5] M. Strik , A. Gonier and P. Williams. *Subsystem Exchange in a Concurrent Design Process Environment*. In Proc. of Design, automation and test in Europe (DATE '08). pages: 953 - 958.

[6] IP-XACT IEEE 1685-2009 Standard. [Online] Available: http://www.accellera.org/

[7] Y. Yankova, G. Kuzmanov, K. Bertels, G. Gaydadjiev, J. Lu and S. Vassiliadis. *DWARV: Delft Workbench Automated Reconfigurable VHDL Generator*, In Field Programmable Logic and Applications (FPL'07).

[8] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, E. Deprettere". *Daedalus: Toward Composable Multimedia MP-SoC Design*, In Proc. of the Design Automation Conference (DAC'08), June 2008.

[9] M. Wirthlin et al., *OpenFPGA CoreLib core library interoperability effort*, In Parallel Computing journal (May 2008), Vol 34, pages: 231-244.

[10] A. Arnesen, N. Rollins and M. Wirthlin, *A Multi-Layered XML Schema and Design Tool For Reusing and Integrating FPGA IP*, In Field Programmable Logic and Applications (FPL '09). pages: 472 - 475

[11] H. Nikolov, T. Stefanov, and E. Deprettere. *Systematic and Automated Multi-processor System Design, Programming, and Implementation*. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol. 27, no. 3, March 2008.

[12] S. van Haastregt and B. Kienhuis. *Automated Synthesis of Streaming C Applications to Process Networks in Hardware*. In Design, Automation and Test in Europe (DATE'09). pages: 890 - 893.

[13] S. Verdoolaege, H. Nikolov and T. Stefanov. *PN: a Tool for Improved Derivation of Process Networks*. In EURASIP Journal on Embedded Systems, vol. 2007, Article ID 75947.

[14] Synphony C Compiler. *PICO Technology*. [Online] Available: http://www.synopsys.com/Systems/BlockDesign/HLS/Pages/SynphonyC-Compiler.aspx