Cloud-based Design and Virtual Prototyping Environment for Embedded Systems

http://dx.doi.org/10.3991/ijoe.v12i09.6142

S. Werner[1], A. Lauber[1], M. Koedam[2], J. Becker[1], E. Sax[1], K. Goossens[2]
[1] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[2] Eindhoven University of Technology (TUE), Eindhoven, Netherlands

***Abstract*—The design and test of Multi-Processor System-on-Chips (MPSoCs) and development of distributed applications and/or operating systems executed on those hardware platforms is one of the biggest challenges in today's system design. This applies in particular when short time-to-market constraints impose serious limitations on the exploration of the design space. The use of virtual platforms can help in decreasing the development and test cycles. In this paper, we present a cloud-based environment supporting the user in designing heterogeneous MPSoCs and developing distributed applications. Therefore, the design environment generates virtual platforms automatically allowing fast prototyping cycles especially in the software development process, and exports the design to a hardware flow synthesizing compatible FPGA designs. The extension of the peripheral models with debug information supports the developer during test and debug cycles and avoids the need of adding special debug codes in the application. This improves the readability, portability and maintainability of produced software. Additionally, this paper presents the benefits of using cloud-based design environments in engineers' trainings and educations. Therefore, the framework supports testing the system including complex software stacks with prerecorded data or testbenches.**

***Index Terms*—Rapid Prototyping, Virtual Platform, Parallel Programming, cloud-based services, OVP, System Level Design, Simulation**

I. INTRODUCTION

The design and test of Multi-Processor System-on-Chips (MPSoCs) including software has proven to be an attractive challenge in embedded systems design automation. The increasingly complex hardware/software-design for those systems, associated with short time-to-market constraints impose serious limitations on the exploration of the design space, and make it necessary to introduce new kinds of development environments and methodologies. The use of virtual platforms and high level simulation may decrease the time-to-market of these architectures while providing the means to exploit, debug and verify architectures with different features and at early development stages.

Several high level simulation frameworks have emerged in the recent past or have been constantly extended (e.g. SystemC [1]) to stay relevant in the continuing trend towards higher abstractions and faster simulations. Instruction Set Simulators (ISS) like Open Virtual Platforms (OVP) [2] focusses on maximized execution speed by using morph functions utilizing binary translation into native host instructions. Thus it allows to test code compiled for the targeted hardware architecture, without the need to recompile it for the real hardware platform. OVP is well supported by several IP vendors and thus offers a selection of already implemented models of processors and peripherals (e.g. timer, UART). As a trade-off for their execution speed, the simulation is only instruction accurate and there is no native way to improve accuracy or bring timing information into the simulations. But virtual prototyping environments like OVP offer in-depths debugging features compared to the targeted platform in hardware.

The rest of the paper is organized as follows: Section II introduces other frameworks for design space exploration and system level design of MPSoCs. Section III gives an overview of the design environment developed during a collaborative research project. In section IV the virtual prototyping platform environment controlled and used by the design environment is explained. Section V and VI describe and evaluate the use of the remote design and prototyping environment by means of two different application examples. Section VII concludes the results and gives an outlook to planned extensions improving the usability and extensibility.

II. RELATED WORK

The design of and application development for heterogeneous MPSoCs is challenging in terms of choosing an architecture matching all constraints and developing a distributed application suitable for the chosen architecture. Therefore, a number of different approaches for design and simulation environments using frameworks for virtual prototyping and/or design space exploration in their back end were developed. SystemC [1] is the most popular framework and offers the possibility to specify and simulate software and hardware blocks of a system at different levels of abstraction. Besides the option of simulation, SystemC code can serve as input for High Level Synthesis (HLS) tools. This allows the development of hardware and software components on one code base and synthesizing the hardware blocks directly to emulate systems on FPGAs. But there are also approaches using Instructions Set Simulators (ISS) like Open Virtual Platforms (OVP) [2].

In [3] Ambrose et al. present a framework for rapid prototyping heterogeneous multicore systems in FPGA. They developed a remote accessible framework named ARGUS to design systems consisting of pre-designed components in hardware and software. For evaluation purposes multiple FPGA boards are set up with a server to allow testing

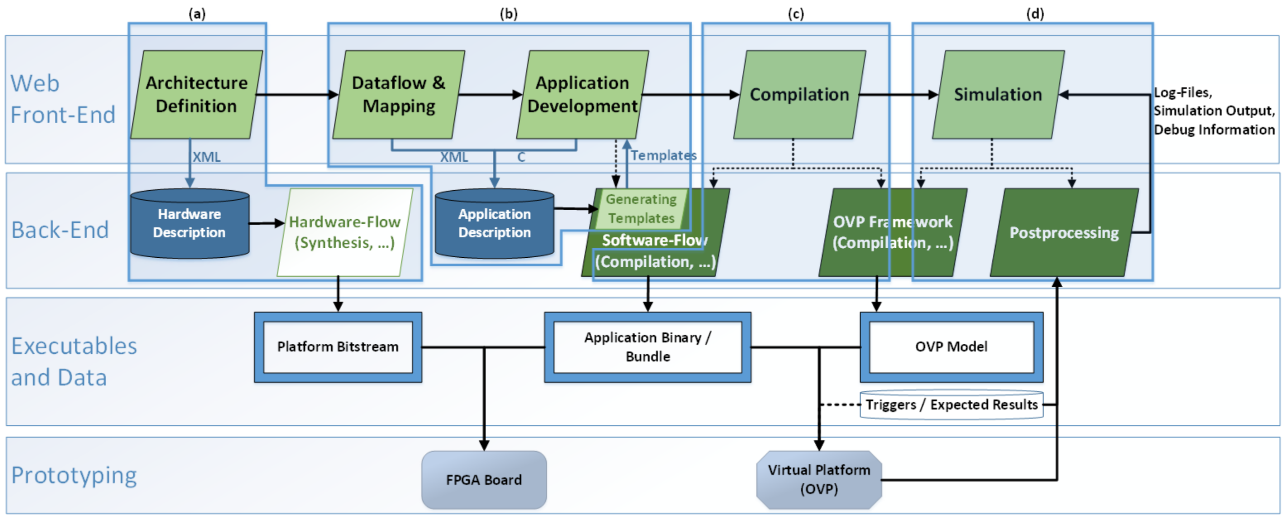


Figure 1. Software Architecture of SimplifyDE and Flow integration

the assembled system remotely. Since the design needs to be synthesized each time before it can be tested, this approach is not suitable for rapid prototyping software and needs real FPGA boards for testing.

The "SystemCoDesigner" developed by Haubelt et al. [4] accelerates the design space exploration using behavioral SystemC models. It provides a correct-by-construction generation of rapid prototypes from behavioral model. The framework requires the SystemC models to be written using the SYSTEMoC library and to only communicate via SystemC FIFOs. Therefore, the application domain is restricted to multi media, networking, and streaming applications. SystemCo-Designer explores the design space with annotated SystemC codes and information from a component library to select an implementation. As in [3] the rapid prototyping of the selected implementation is done on a real FPGA board, which causes the overhead of synthesizing before prototyping.

Zimmermann et al. [5] present a simulation-oriented framework to overcome this bottleneck. It uses UML-based descriptions of software and hardware architecture to generate a virtual execution platform in SystemC. The hardware platform is specified and configured using SysML and equipped with additional information. The software components are wrapped and instantiated inside the hardware resources. To link the several system components the framework uses a layered approach for TLM-based simulation. This approach needs to annotate software codes and runs the software on the host. Therefore, it has limitations in the development of hardware-related software stacks. A way to reuse embedded hardware and software components to improve the system design process are outlined in [6].

In [7] Abdi et al. present an "Embedded System Environment" (ESE) focusing more on the application development. It consists of a set of tools supporting a model-based design methodology for MPSoCs and provides two levels of abstraction levels. The first one is the ESE Front End and provides the automatic generation of TLM-code (and virtual platforms) and application code. Thus, it supports the software development process by providing Software-in-the-Loop in TLM-level. The ESE Back End represents the second level of abstraction. It automatically transforms the TLM description of the Front End to a CAM model (Cycle Accurate Model), and creates and synthesizes FPGA project files using Xilinx tools. Since the ESE generates SystemC codes using Timed TLM (TTLM) and operating systems and related codes are modeled as sc_modules, the software stacks are executed on the host natively and not on an emulated or simulated processor model. This improves the performance of the simulation but does not allow the development of low level software stacks due to the incompatibility between the binary interfaces of the host and the target platform.

To overcome the lack of missing ABI compatibility Aguiar et al. [8] uses an instruction set simulator (ISS) in their Hellfire Framework. The framework allows the application development for Hellfire OS. It creates a simulation platform for a homogeneous MPSoC, runs simulations and provides analysis facilities even for the operating system. If one of the developed C applications needs to run on several cores, the user has to split the code in several tasks manually. Magalhaes et al. [9] extended the Hellfire Framework with web-based interface to generate the simulation platform and add NoC support to the simulator.

Almeida et al. [10] introduce a cloud based framework using virtual platforms and functional simulation. Setting up and compiling virtual platforms allows significantly shorter cycles in software development and design space exploration compared to synthesizing FPGA designs. Another advantage is the missing limitation in terms of hardware resources. This regards the number of simultaneously available boards, but also the constraints given by the available FPGA like size of block memory (BRAM). But the proposed framework only allows the design and exploration of bus based multiprocessor architectures running small independently working baremetal software stacks.

III. THE DESIGN ENVIRONMENT

The presented design environment, called SimplifyDE, was developed during a collaborative research project. Its original intention is giving the terminal and text based simulation framework Open Virtual Platforms (OVP) a graphical user interface (GUI) to increase the productivity. It allows the user to start quickly with setting up new virtual platforms for a targeted hardware architecture and performing simulations, without the need of a long training period. Besides the original focus of providing a web-based GUI for OVP, SimplifyDE is integrated in the hardware and software flow as shown in Figure 1, targeting the FlexTiles Development Platform [11] that is based on the CompSOC design flow [12] including support for the realtime operating system CoMik [13] and its communication services. Thus, SimplifyDE can generate and export XML-files which describe the hardware and are compatible to the hardware flow and can be used to synthesize an FPGA design directly (see (a) in Figure 1). Furthermore, SimplifyDE is integrated in the software flow [12] developed for the operating system to generate code templates containing all management code and hardware abstraction layers (block (b) in Figure 1). This hides the complexity of this code from the user who only needs to insert own functions as described later in section B of this chapter. During the "Compilation" step (c) the virtual prototyping platform is setup and the "OVP Model" compiled, and the application codes including the operating system and drivers are cross-compiled and the "Application Binary/Bundle" is generated. The final "Simulation" step (d) runs the simulation, loads the cross-compiled executables in the virtual platform and processes the simulation output. The postprocessing is optional and depends on the chosen test method.

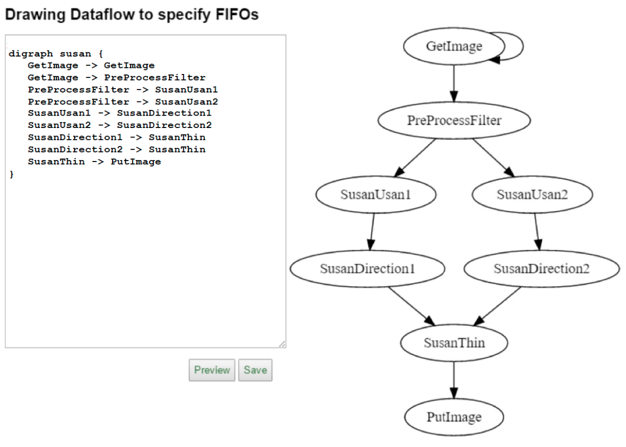
A. Designing an architecture

With the dialog shown in Figure 3 the user can specify the system view of the targeted hardware architecture. The depicted example consists of five MicroBlaze processors, one MB Monitor (explained later in section V.B), a shared memory and a DVI card. The components are interconnected with a Network-on-Chip in mesh topology and a dimension of 3 by 2.

New system components can be added by drag-and-drop one of the available components in the menu bar (white bar) in an empty square in the schematic representation. To remove or configure a component the user can use a context menu opened by right-clicking on the module. The user can also define hybrid architectures using NoCs and busses for communication. For both kinds of interconnect, options for the transmission frequency and token size, defining the corresponding parameters for the communication infrastructure, are available.

Additionally, the system components like processor systems and video card can be configured. For instance, the user can select if the processor runs a baremetal application or an operating system, in the presented work different versions of CoMik. Furthermore, settings for the clock frequency and memory size can be made. For the supported processor system components the user can specify which peripheral devices like AD-converters, PWMs and UART are part of the CPU system and the corresponding memory map. For input and output devices the user can upload and assign files which can be used during the prototyping phase, serving as input data to be processed by the system component or as reference values for data calculated by it respectively ("Postprocessing" in Figure 1).

The designed architecture is not only used to create a corresponding virtual platform. It can also generate and export an XML file compatible to the hardware design flow using Xilinx tools. This file contains the description of the hardware system design generated in the GUI and is


Drawing Dataflow to specify FIFOs

```
digraph susan {
    GetImage -> GetImage
    GetImage -> PreProcessFilter
    PreProcessFilter -> SusanUsan1
    PreProcessFilter -> SusanUsan2
    SusanUsan1 -> SusanDirection1
    SusanUsan2 -> SusanDirection2
    SusanDirection1 -> SusanThin
    SusanDirection2 -> SusanThin
    SusanThin -> PutImage
}
```

Figure 2. Defining a CSDF-graph with dot

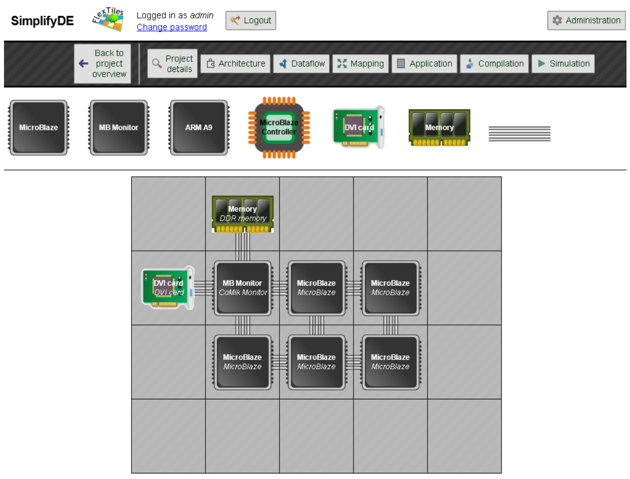


Figure 3. Defining a hardware architecture

compatible to the hardware flow. Thus, the file can be used as input for this part of the FlexTiles toolchain to generate and synthesize the hardware platform for the Xilinx ML-605 or FlexTiles Development Platform [11].

B. Application design

The presented framework is optimized to support the development of streaming applications based on the Model-of-Computation (MoC) "Cyclo-Static Dataflow" (CSDF) [14]. The CSDF MoC is suited for streaming applications like image processing where there is an end-to-end requirement for throughput and latency. The actors in a CSDF-graph represent the computational kernels, and the edges represent the data transfer between the actors. The actors and edges can be manual, or automatically mapped to the MPSoC platform.

In the framework the user can draw the highlevel structure of the application by defining a CSDF-graph for the application to be implemented using the dot format. A preview of the graph corresponding to the current description is drawn and shown to its right as depicted in Figure 2. After saving the dataflow graph, the information can be used in the mapping dialog shown in Figure 4. Here the nodes of the CSDF-graph are represented by red squares. With the list boxes at the bottom edges the corresponding node can be assigned to one of the processing system components available in the previously defined hardware system architecture (Figure 3).

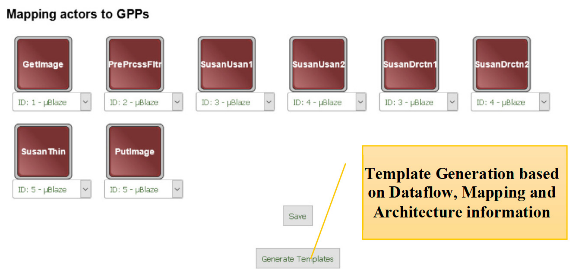


Figure 4. Mapping nodes of CSDF-graph to processing units (some names are shorten improving readability)

After the definition of the basic architecture of the streaming application and its initial mapping to the architecture, the user can create and assign C source files to the processing units and define common files, used by all processing units. The text editor integrated in the framework supports syntax highlighting and can be used to either develop C code from scratch or inserting available resources by uploading or pasting them in the source files. To ease the development of CSDF and CoMik related applications the user can “Generate Templates”. Doing so, the information given by the user in the dialogs shown in Figure 2 and Figure 4 is used to generate C source files containing all operating system, hardware drivers and communication management code. After the generation process the user only needs to insert the user code for the actors, represented by the nodes of the CSDF graph in the corresponding function prototypes named as the nodes in the graph.

The implemented code can be cross compiled within the cloud based design environment. If the compiler produces errors or warnings, these will be shown in the browser. After the cross compilation the generated executables can be directly executed in the framework on a virtual platform based on the definition of the system architecture or downloaded to run the software on hands-on hardware devices. The virtual platform and the simulation environment used in the framework are introduced and explained in the next chapter.

IV. SIMULATION ENVIRONMENT

A. *Open Virtual Platforms*

Compared to SystemC OVP focusses on maximized execution speed by using morph functions utilizing binary translation into native host instructions. This allows to test software by using the application binaries already cross-compiled for the target architecture without the need of additional annotations in the code. Therefore, the tested applications are completely compatible to the Application Binary Interface (ABI) of the target system. Since several IP vendors support OVP and a large selection of already implemented models of processors and peripherals (e.g. timer, UART, etc.) is available. As a trade-off for the execution speed, the simulation is only instruction accurate and there is no native way to improve accuracy or bring timing information into the simulations. On the other side, OVP allows in-depths debugging of the created platforms. Thus, GDB can be used to debug all parts of the system, not only the software running on the CPU and the register content of the CPU as on a standard embedded hardware platform. Additionally, it can be used to get information out of the peripheral devices without the need of specialized tools. So the users do not need an external programmer to debug.

Furthermore, the virtual platform simulated in OVP executes the code already cross-compiled for the targeted architecture. This means that no recompilation is needed to use it later on the real hardware platform. An executable successfully tested on a virtual platform fulfills all functional requirements needed to run on the targeted hardware platform.

```
PWM Channel 0 disabled
PWM:: Timestep 0: CPRD1 written to 0. This equals a
PWM-frequency of 0 Hz
PWM:: Timestep 0: CPRD1 written to 444. This equals a
PWM-frequency of 18018 Hz
PWM Channel 0 enabled
```

Listing 1.: Output of the Simulation environment showing enhanced debugging output of the peripherals

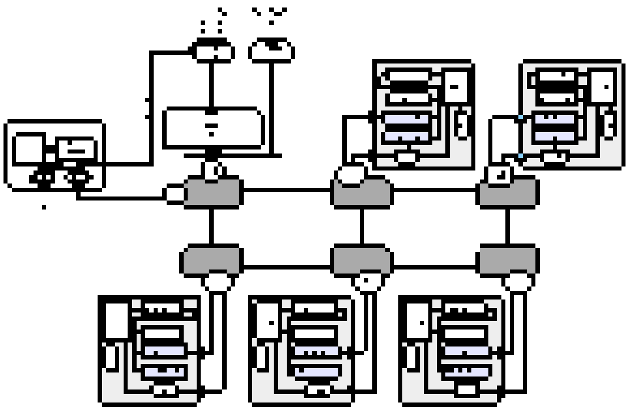


Figure 5. block diagram of the targeted hardware architecture

B. *Enhanced debugging features compared to real hardware*

A big challenge in embedded parallel programming is the reduced availability of debugging features compared to tools like Visual Studio used for developing applications for desktop systems since usually special debug hardware is needed like the MDM available for MicroBlaze processors on Xilinx devices. To open the black box, most embedded systems represent, the models of the peripherals used in the virtual platform are extended with some intelligence and knowledge taken from the data sheets like the right order of setting bits in the control registers of the timer for example. Listing 1 shows an example output of the application scenario introduced in VI.C where the PWM peripheral gives information about the currently set frequency and channel for controlling electrical engines. Other messages give information about the validity of set values or compliance with values range. So the peripheral models can give a hint to the user if the code does not fulfill the specification. This is of special interest for typical issues occurring when programming embedded systems like potentially incompatible data caused by different endiannesses or different handling of signs, causing in overflows for example. The biggest advantage of introducing this kind debug information in the peripheral models of the virtual platform is that there is no need any more to insert debug sequences and outputs in the application code. This reduces the overhead during the development process since the application designer can focus just on application programming and does not need deep knowledge about the peripheral devices. Furthermore, the performance and maintainability of the source codes are improved since the number of lines of code can be reduced significantly. This is especially the case, if one application is developed for several different architectures which typically leads to the use of complex ifdef-constructs.

V. APPLICATION DEVELOPMENT – SUSAN

A. *The SUSAN application*

SUSAN [15] ("Smallest Univalue Segment Assimilating Nucleus") is a collection of algorithms performing edge and corner detection. The SUSAN application, as depicted in Figure 6 has been extended with a small contrast enhance filter and is further parallelized to meet the required throughput constraints.

The algorithms determine which parts of the image are closely related to one pixel, by associating each individual pixel with a local image region which is of similar brightness to that pixel. Doing so, a mask is applied centered in the pixel of interest. The amount of pixels within the mask which have a similar intensity is called "Univalue Segment Assimilating Nucleus" (USAN). Those pixels that have a USAN smaller than a predefined threshold are good candidates for being edge points. If the number of pixels within a USAN is smaller than some (predefined) threshold, there is a possibility that the pixel of interest is an edge point. Afterwards the momentums of the USAN are computed to determine the direction of the edge. A following thinning removes unwanted edge pointes and adds edge points where they should be reported but have not been.

The described methodology is split into tasks as depicted in Figure 6. GetImage reads in a colored image and converts it to grayscale (brightness). Furthermore, this task enables data parallelism within an image, the image is chopped in sections called Minimum Coded Units (MCUs). Each MCU is transmitted as so-called MCU_Block. The following task SUSANUsan relays the MCU_Block to SUSANDirection after processing each image MCU, together with the calculated MCU_EdgeStrength. Additionally it calculates a Brightness Lookup Table (BLT) and keeps it locally. To avoid communication overhead the task SUSANDirection computes a locally stored BLT as well and calculates MCU_EdgeDirection and relays it to the next task together with MCU_Block and MCU_EdgeStrength. SUSANThin thins the individual edges and transfers the MCU_Block and the resulting edges to PutImage. This task finally stitches the MCUs together and draws the identified edge(s) on the output image.

B. *Targeted hardware platform and virtual equivalent*

The targeted architecture of the multi-processor layer of the hardware is depicted in Figure 5 and is an instance of the FlexTiles Development Platform [11][12]. This design flow provides a tile based template for creating SoCs for running applications with mixed time-criticality. It relies on two complexity-reducing concepts: composability and predictability to reduce the design and integration complexity of applications. The hardware platform consists of different types of tiles connected to each other via a Network-On-Chip (NoC) [16]. The NoC, at runtime, can be configured to provide isolated point-to-point (ptp) connec-

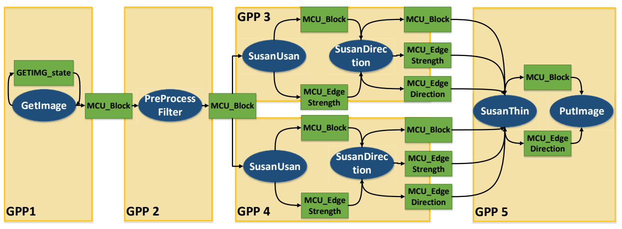


Figure 6. SDF graph of SUSAN Edge Detection

tions with a guaranteed throughput and latency. In this instance there are five MicroBlaze based processor tiles used for computation, each tile runs the real-time operating system CoMik[13]. These tiles communication over the ptp connections using a software FIFO implementation and direct memory access (DMA) units. Because the FlexTiles platform uses distributed shared memory hierarchy, the NoC can be replaced by direct memory access in OVP without violating the models correctness in OVP. In the OVP model the processor tiles use the processor model of the Microblaze and an identical memory hierarchy. The DMA is modeled using an ABI compatible module and the timer unit was replaced by a model of the Xilinx timer extended with additional interrupt signals.

The Monitor tile is used for the initial bootstrap of the platform and after the platform is started it gathers all debug outputs from the different processor tiles via dedicated debug FIFOs. It sends these messages as (compressed) binary information via UART to a host machine. The OVP model exactly models this setup (see "MB Monitor" in Figure 3).

The DDR tile consists of an Ethernet interface to transfer data, like application bundles, from outside the platform to the DDR. Application bundles contain a description of the required resources and the executable code. These bundles once placed at a specific memory location are processed by the multi-tile loader [17] that instantiates the new application in the running system. In the simulation model, data can be directly placed into the shared memory. Therefore, the Ethernet interface is not needed to load data in the simulation environment. Thus, it is not part of the simulated hardware platform.

An extension card providing DVI interfaces for streaming video input and output is connected to the FlexTiles Development Platform. It has a high-performance direct interface to the DDR memory, so that the video buffers can be located in the DDR. The framebuffers can then be directly accessed by the processor tiles connected via the NoC. The initial configuration of the DVI interfaces like setting the framebuffer location is done by the Monitor tile. To support video input and video output in the simulation model as well, it was updated with some additional peripherals, which are described in the next sections.

C. *Video peripherals in the Virtual Platform*

For a first validation of the SUSAN edge detection running on our virtual platform we configured the video peripheral in file mode. This enables using image files as input and storing the calculated image as file as well. In both cases the peripherals appear as Xilinx VGA-/DVI-IP-core and the same driver code can be used as on the real hardware platform. The validation with image files has the advantage of reproducible results in case of misbehavior. Furthermore, it is possible to stop the execution of the application and use debugging tools like GDB, which is not possible when a camera delivers a continuous input stream. Since the OVP model potentially runs faster than the physical FPGA prototype, this approach has another advantage: The next image can be read in directly after the calculation for the previous image has finished. This avoids slowing down the simulation to be compatible to the frame rate of an incoming stream provided by a camera, and therefore shortens the test cycles in the early phase of the validation process. After the successful validation of the implemented SUSAN edge detection handling files, the peripherals are adapted to access hardware devices connected to the host machine as described in [18]. This validates the output of the application running in the simulated environment with the output of the same application running on an FPGA-board by simply comparing the images shown in the SDL window ("connected" to OVP platform) and in the screen (connected to FPGA) respectively.

D. *Results*

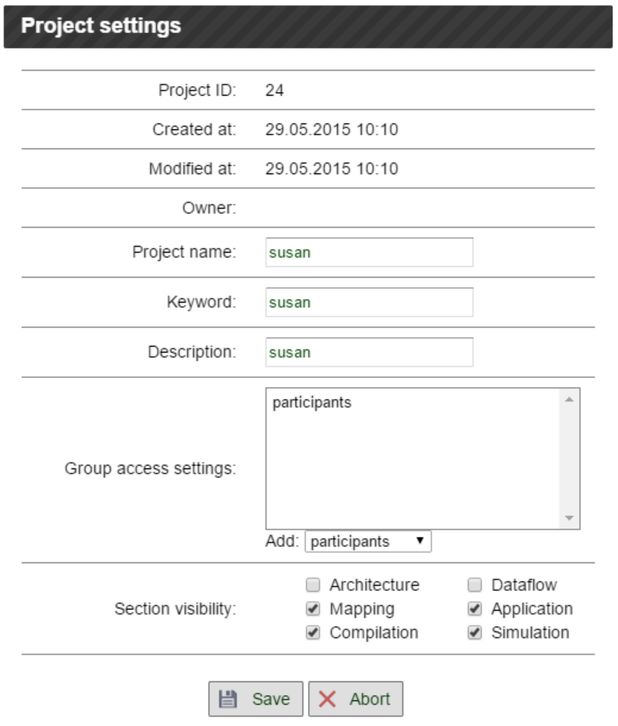
The SUSAN application was used to validate the complete design environment. Using SimplifyDE the application was ported to the target platform and validated for functional correctness. In a second step modification where made to the application so it efficiently uses the chosen hardware platform. Because this part of the application design could be executed on the generated OVP platform model of the hardware it allowed for quick iterations and verification of the changes like; parallelizing part of the application, increasing block sizes and the addition of a contrast enhance filter. Once the result was satisfactory the generated application bundle was ran on the synthesized version of the hardware on the FlexTiles FPGA board, producing the desired output. Important to note is that the applications running on the OVP platform and the final FPGA prototype are completely identical and do not require recompilation. Even the underlying operating system modifications where limited to add support for a different timer used on OVP.

VI. EATURES SUPPORTING ENGINEERS' TRAININGS AND COLLABORATIONS

Besides the features described in the previous chapter which support system designers and application developers in designing and programming MPSoCs, we extended SimplifyDE with some features focusing on engineers' training and collaboration, described in the subsection A. The advantages of web based remote physical labs and the reasons for improved understanding of modeling and programming can be found in [19]. The educational benefits of using a virtual IDE for programming is discussed in [20] and [21]. The section B presents how to extend design projects with unit tests, which allows using the framework for first experience and education purposes, as well as for collaborative work on complex MPSoC applications. The final subsection C gives a short overview about collected experience in using the design environment in workshops and largescale trainings. An overview of remote laboratory for embedded systems design and the advantage of hardware-software integration and testing is given in [22].

A. *User management and project protection*

To allow collaboration on one hand and provide the opportunity of using the framework in trainings for MPSoC programming, a specialized user management system is integrated in SimplifyDE. It allows the definition of groups and users, where one user can be assigned to several groups. To ease the creation of user accounts and the group assignments by automation, a CSV file can be imported. This feature is of special interest when the framework is used in largescale trainings as described later in section C.



Project settings

Project ID:	24
Created at:	29.05.2015 10:10
Modified at:	29.05.2015 10:10
Owner:	
Project name:	susan
Keyword:	susan
Description:	susan
Group access settings:	participants Add: participants
Section visibility:	☐ Architecture ☐ Dataflow ✓ Mapping ✓ Application ✓ Compilation ✓ Simulation

Figure 7. Access rights in projects settings

The access rights specifies the use of the particular parts of the flow, described in chapter III, for groups of users. The creator of a design project can specify these settings for each project, as shown in Figure 7. The depicted configuration for the project "susan" allows only the group "participants" to use the project. Only users of groups listed in "Group access settings" can copy the project for further processing, here the group "participant". However, even in their own copy they cannot change the defined hardware architecture and dataflow. As can be seen, the different "visibility" setting correspond to the particular steps of the flow illustrated in Figure 1. For the example shown it means that the users cannot use part (a) and the first half of part (b). One real-world scenario for those restrictions is the use for training purposes. An expert can create an example project and provide them to participants of a workshop about programming of MPSoCs. In this example the participants can work with the provided source codes and change the mapping of tasks to the processors. If several users need to work on one project with full access rights, they must be members of the same group.

B. *Enhance integrated projects with unit tests*

Another feature the design environment provides is the opportunity to support training sessions for application developers with standard sample solutions. Here the owner of an integrated project can deliver a project including all codes to other users while some codes are only available in the background and not visible to the users. This is possible due to the sequence the Web Front-End process files to compile the applications and the virtual platform, illustrated in Figure 8.

Each time the Web Front-End triggers operations processing files, it checks if a temporary folder for this user session already exists. If this is not the case, it copies a template folder containing all folder structures, Makefiles and shared libraries needed by the Back-End to perform all operations. Afterwards, the sequence generates a list of all files created during the "Application Development" step (Figure 1, part (b)). Since files uploaded, created or edited during this step are only stored in the database of the SimplifyDE, the framework iterates over all files and stores them in the private copy of the template folder if a non-empty version is available in the database. Therefore, empty files are not loaded from the database and potentially existing files in the private template folder are not replaced. Since the WebGUI only shows information stored in the database, this constraint allows the project owner to provide source files which are not visible to users who copied the project. Furthermore, the users can develop their own codes within the design environment and replace the codes partially.

During workshops and training sessions this feature can accelerate achieving learning results, since it allows to test single modules of a complexer application very early. It allows the participants to test their particular modules (i.e. for controlling ADCs to read in sensor data), even if they have not implemented the entire software yet. Missing source files are replaced by the framework during the cross-compilation process. In training scenarios where several persons should work collaborative on a bigger project, this feature allows the participants to test their modules independently before integrating them in one bigger software system.

C. *Use in engineers' trainings and collaborative work*

The design and simulation environment was used in several trainings and workshops on conferences to present the benefits achieved by using virtual platforms for rapid prototyping and to give hands-on experiences to participants. Usually those workshops have a limited number of participants. To highlight the benefits of providing SimplifyDE as web-based service, this section gives a brief overview over a project-based training and focuses on the use of this framework in a largescale training with several sessions where participants need to work collaborative. A complete description of the course and teaching session and its procedure can be found in [23]. Other approaches to overcome these limitations by using a learning management system can be found in [24] and [25].

At our university we provide hardware related programming laboratories for the faculties of electrical engineering and mechanical engineering, with up to 350 students participating. In one of the laboratories at the faculty of electrical engineering and information technologies the students have to program the hardware abstraction layer (HAL) of a two-wheeled, self-balancing vehicle. The huge amount of students leads to a bottleneck in programmable hardware platforms, since only a limited number of hardware platforms can be provided. However, access to the platform during programming is needed for debugging, especially for hardware related programming. Therefore, a virtual platform is set up, that can be accessed at any time

```
timestep: 0.........
<...>
 timestep: 10000......
*** simulation finished ***
<...>
Info
Info -------------------------------------------------
Info SIMULATION TIME STATISTICS
Info   Simulated time        : 105.35 seconds
Info   User time             : 2.89 seconds
Info   System time           : 0.05 seconds
Info   Elapsed time          : 2.95 seconds
Info   Real time ratio       : 35.77x faster
Info -------------------------------------------------
```

Listing 2: Output of the Simulation environment showing the speedup compared to real hardware

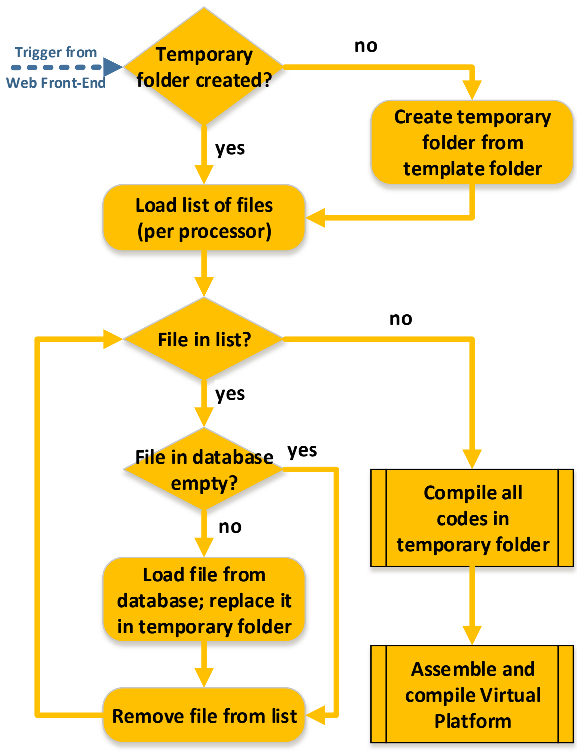

Trigger from Web Front-End
Temporary folder created?
no
Create temporary folder from template folder
yes
Load list of files (per processor)
File in list?
no
yes
File in database empty?
yes
no
Compile all codes in temporary folder
Load file from database; replace it in temporary folder
Assemble and compile Virtual Platform
Remove file from list

Figure 8. Sequence of the compilation process

and any place for debugging during the laboratories and home programming. On the other hand, our virtual platform provides more options in terms of debugging than the real hardware.

Listing 2 shows one of the biggest advantages the use of a virtual platform has in our scenario. The models of the input and output devices, in this scenario the ADCs and PWMs, can run with significantly higher sample rates than possible in the real hardware platform. This allows to simulate a test drive, needing about 105 seconds in reality, in less than 3 seconds. So the use of the virtual platform not only avoids the bottleneck of insufficient number of available hardware devices. It allows much shorter testing and debugging cycles.

Further acceleration of testing is provided by the option of testing single modules in the framework. It allows the participants to test their particular modules (i.e. for controlling the ADCs), even if they have not implemented the entire software yet. Source files, not yet implemented, are replaced by the framework during the cross-compilation process. This allows the students to test their modules independently before integrating them in one software system. Additionally, the prototyping environment delivers four different outputs after postprocessing the log-information: the standard output of the compiler showing potential warnings and error, an output of the simulator showing the debugging hints of the peripheral models and some more information about the overall simulation run. The other two outputs compare the data pre-recorded during test drives with the simulated values for the PWM and the GPIOs including information about the standard deviation. This allows to check quickly if the values calculated by the students' codes are correct. A complete description of the hands-on experience in the laboratory and the usage of the simulator as well as test phases and evaluation can be found in Werner et al. [26]. After a successful simulation the executable can be downloaded and used to program the real hardware platform.

VII. CONCLUSION AND ONGOING WORK

This paper presents the cloud-based design environment SimplifyDE providing an intuitive GUI for designing system architectures for virtual and FPGA-based single and multi-processor designs. Besides the option of generating files compatible to the Xilinx toolflow to support the synthesis of FPGA designs directly, the design environment generates a vitual platform based on Open Virtual Platforms (OVP) which is fully compatible to the FPGA design in terms of software and application development. Thus, SimplifyDE supports the cross-platform development of embedded control applications including operating systems, drivers and other complex software stacks within the browser. Additionally, with CSDF a widely used Model of Computation for streaming applications is supported by generating C template files, where the user only needs to add code executed by the actors. The code responsible for handling and controlling the underlying hardware is hidden from the developer. The entire source code can be compiled, tested and debugged inside the cloud-based IDE. In doing so, the models of the peripheral devices used in the virtual platform support the user with further information about the internal states of the peripherals which opens the black box, the physical embedded system usually represents. This helps shorten the development cycles and keep design complexity under control.

Secondary, this paper presents the successful transfer of results obtained in a collaborative research project (SimplifyDE and the ABI-compatible simulation based on OVP) in engineers' training by using the framework in workshops for advanced users and in a hardware-related programming laboratory participated by more than 300 undergraduate students in electrical and mechtronic engineering. Especially the students at our institute use the rapid prototyping features of the framework extensively. Thereby, they benefit from the enhanced debugging features of the virtual platform improving their learning curve and its fast execution, resulting in shorten test cycles.

The work on SimplifyDE is still on-going. We will integrate some more assistance features improving the support in defining unit tests in own projects. Another task is extending the back end layer of the simulation environment with other simulation frameworks like SystemC and Simulink to provide a fully integrated co-simulation environment.

REFERENCES

[1] SystemC. [Online]. Available: http://www.accellera.org/home/

[2] Open virtual platform. [Online]. Available: www.ovpworld.org

[3] Ambrose, J.A.; Tuo Li; Murphy, D.; Gargg, S.; Higgins, N.; Parameswaran, S., "ARGUS: A Framework for Rapid Design and Prototype of Heterogeneous Multicore Systems in FPGA," in VLSI Design (VLSID), 2015 28th International Conference on , vol., no., pp.29-34, 3-7 Jan. 2015, http://dx.doi.org/10.1109/VLSID.2015.10

[4] C. Haubelt, T. Schlichter, J. Keinert and M. Meredith, "SystemCoDesigner: Automatic design space exploration and rapid prototyping from behavioral models," Design Automation Conference, DAC 2008. 45th ACM/IEEE, Anaheim, CA, 2008. http://dx.doi.org/10.1145/1391469.1391616

[5] J. Zimmermann, S. Stattelmann, A. Viehl, O. Bringmann and W. Rosenstiel, "Model-driven virtual prototyping for real-time simulation of distributed embedded systems," 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12), Karlsruhe, 2012, pp. 201-210. http://dx.doi.org/10.1109/SIES.2012.6356586

[6] A. Parkhomenko, A. Sokolyanskii, V. Shepelenko, Y. Zalyubovskiy and O. Gladkova, "Reusable solutions for embedded systems design," 2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV), Madrid, 2016. http://dx.doi.org/10.1109/rev.2016.7444491

[7] S. Abdi, Y. Hwang, L. Yu, H. Cho, I. Viskic and D. D. Gajski, "Embedded system environment: A framework for TLM-based design and prototyping," Proceedings of 21st IEEE International Symposium on Rapid System Protyping, Fairfax, VA, 2010. http://dx.doi.org/10.1109/RSP.2010.5656342

[8] A. Aguiar and F. Hessel, "Virtual Hellfire Hypervisor: Extending Hellfire Framework for embedded virtualization support," Quality Electronic Design (ISQED), 2011 12th International Symposium on, Santa Clara, CA, 2011, pp. 1-8. http://dx.doi.org/10.1109/ISQED.2011.5770725

[9] F. G. Magalhaes, O. Longhi, S. J. Filho, A. Aguiar and F. Hessel, "NoC-based platform for embedded software design: An extension of the Hellfire Framework," Thirteenth International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, 2012. http://dx.doi.org/10.1109/ISQED.2012.6187480

[10] Marchesan Almeida, G.; Bellaver Longhi, O.; Bruckschloegl, T.; Hubner, M.; Hessel, F.; Becker, J., "Simplify: A Framework for Enabling Fast Functional/Behavioral Validation of Multiprocessor Architectures in the Cloud," in Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International , vol., no., pp.2200-2205, 20-24 May 2013,

[11] SMT166 – Development Board, version 1.01 (8th February 2013), Sundance, www.sundance.com/prod_info.php?board=smt166

[12] Sven Goossens, Benny Akesson, Ashkan Beyranvand Nejad, Andrew Nelson, Martijn Koedam, and Kees Goossens, The CompSOC design flow for virtual execution platforms; Invited paper, in Proc. FPGA World, September 2013.

[13] Nelson, A.; Nejad, A.B.; Molnos, A.; Koedam, M.; Goossens, K., "CoMik: A predictable and cycle-accurately composable real-time microkernel," Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014 , vol., no., pp.1,4, 24-28 March 2014

[14] G. Bilsen et al., "Cycle-static dataflow," IEEE Trans. Signal Process., vol. 44, no. 2, pp. 397–408, 1996. http://dx.doi.org/10.1109/78.485935

[15] S. M. Smith, J. M. Brady; "SUSAN - A New Approach to Low Level Image Processing", International Journal of Computer Vision (Vol. 23, 1995, p. 45-78); DOI: 10.1.1.24.2763

[16] Radu Stefan, Anca Molnos, and Kees Goossens, dAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up. In IEEE Transactions on Computers, 2012.

[17] Shubhendu Sinha, Martijn Koedam, Gabriela Breaban, Andrew Nelson, Ashkan Nejad, Marc Geilen, Kees Goossens, Composable and Predictable Dynamic Loading for Time-Critical Partitioned Systems on Multiprocessor Architectures, Elsevier J. on Microprocessors and Microsystems (MICPRO), November 2015.

[18] S. Werner, L. Masing, F. Lesniak and J. Becker, "Software-in-the-Loop simulation of embedded control applications based on Virtual Platforms," 2015 25th International Conference on Field Programmable Logic and Applications (FPL), London, 2015, http://dx.doi.org/10.1109/fpl.2015.7294020

[19] S. Peter, F. Momtaz and T. Givargis, "From the browser to the remote physical lab: Programming cyber-physical systems," Frontiers in Education Conference (FIE), IEEE, El Paso, TX, 2015. http://dx.doi.org/10.1109/fie.2015.7344228

[20] D. Pawelczak and A. Baumann, "Virtual-C - a programming environment for teaching C in undergraduate programming courses," 2014 IEEE Global Engineering Education Conference (EDUCON), Istanbul, 2014, pp. 1142-1148 http://dx.doi.org/10.1109/EDUCON.2014.7096836

[21] M. Cooper, "Remote laboratories in teaching and learning – issues impinging on widespread adoption in science and engineering education" , International Journal of Online Engineering (iJOE), Vol 1 - No 1, 2005

[22] A. V. Parkhomenko, O. Gladkova, E. Ivanov, A. Sokolyanskii, S. Kurson, "Development and Application of Remote Laboratory for Embedded Systems Design", International Journal of Online Engineering (iJOE), Vol 11 - No 3, 2015

[23] Tradowsky, C.; Lauber, A.; Werner, S.; Beuth, T.; Mueller-Glaser, K. D.; Sax, E., "Porter for the ITIV LABS – Objective-Related Engineering Education in an Undergraduate Laboratory", in Journal of Teaching and Education, vol. 4, pp. 45–58, 2015

[24] T. Richter, P. Grube and D. Boehringer, "Integrating an online programming lab into ILIAS," Remote Engineering and Virtual Instrumentation (REV), 2014 11th International Conference on, Porto, 2014, pp. 31-34; http://dx.doi.org/10.1109/REV.2014.6784187

[25] T. Richter and D. Boehringer, "ViPLab ??? An online programming lab," 2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV), Madrid, 2016, pp. 269-270. http://dx.doi.org/10.1109/rev.2016.7444479

[26] S. Werner, A. Lauber, J. Becker, E. Sax, „Cloud-based Remote Virtual Prototyping Platform for Embedded Control Applications", In 13th International Conference on Remote Engineering and Virtual Instrumentation (REV), 2016 http://dx.doi.org/10.1109/REV.2016.7444459

AUTHORS

S. Werner is with Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

A. Lauber is with Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

M. Koedam is with Eindhoven University of Technology (TUE), Eindhoven, Netherlands.

J. Becker is with Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

E. Sax is with Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

K. Goossens is with Eindhoven University of Technology (TUE), Eindhoven, Netherlands.

Submitted 12 August 2016. Published as resubmitted by the authors 14 September 2016.