Automatic Exploration of Hardware/Software Partitioning

Syed Abbas Ali Shah, Sven Alexander Horsinka, Basitan Farkas, Rolf Meyer, Malden Berekovic
E.I.S., TU Braunschweig, D-38106 Braunschweig, Germany
Email: {shah, shorsinka, farkas, meyer, berekovic}@c3e.cs.tu-bs.de

***Abstract*-Today's complex applications require an efficient hardware/software co-design. Exploring possible hardware/software partitioning scenarios is a time consuming and challenging job. In this work, we present a flexible and automated workflow "SW2TLM" for generating system level hardware accelerators from software applications and their integration to virtual platform framework. SW2TLM enables efficient exploration of different partitioning scenarios with minimal effort, all complexity of the hardware architecture and software integration is hidden from the user. The analysis results captured by the virtual platform will support the decision making towards a final hardware/software partitioning and thereby improve and accelerate the subsequent development process.**

I. INTRODUCTION

Current trends in processing architecture transitions from general purpose to more specialized approaches to achieve higher performance at lower power consumption. Modern System-on-Chips are comprised of several co-processors or accelerators to offload compute intensive tasks from the central processing cores. This trend comes with serious engineering challenges for hardware, software as well as system engineers alike. Applying an established design flow with static hardware/software partitioning from early design stages on is becoming increasingly difficult. Performing the partitioning without evaluating the achievable performance gains can result in poor resource utilization, subpar performance or costly retargeting of the architecture in later design stages.

To address these challenges, virtual platforms and rapid prototyping frameworks using SystemC/TLM2.0 are growing in popularity. These platforms are providing the designers with an executable prototype of the hardware well before any actual hardware becomes available. However, exploring different hardware/software partitioning using these virtual platform frameworks or hardware emulators remains a mostly manual and labour intensive task. Developing hardware representations using transaction level modelling (TLM) techniques for a corresponding section of software can be challenging for developers without a background in computer engineering.

Herein lies the contribution of this work. We present a workflow with a high degree of automation to explore different hardware/software partitions using a SystemC/TLM2.0 virtual platform. Based on sparse configuration input, the presented tool can generate a TLM hardware accelerator based on a specific software functions. Subsequently, it is annotated with timing and power consumption parameters derived from a high-level synthesis based tool-flow and integrated into the target hardware and software environment. From the designers' perspective, only the piece of software needs to be characterized to explore the achievable speedup and energy savings when offloading it to a specific accelerator.

This approach is demonstrated based on the SoCRocket [1] virtual platform and its powerful scripting interface [2] to automate the collection and pre-processing of the captured performance and power data.

The remainder of the paper is structured as follows: The next section reviews the related work in the domain of automatic generation of abstract hardware models. Section III gives additional motivation, why proper hardware/software partitioning is important as well as how performance and power are recorded on the system level. Section IV illustrates the SW2TLM design flow and tool-chain. Section V evaluates the automated framework against three applications from the image processing domain. Section VI concludes and highlights the direction of future work.

II. RELATED WORK

The tool-supported translation of software sections to TLM models is investigated in academia as well with commercial tools. Especially computationally demanding fields steer towards higher specialization of hardware and software architecture. Computer vision is a typical domain in need for acceleration to analyse significant amounts data at a high rate. Mefenza et al. [3] presented a rapid prototyping framework to evaluate different partitioning and mapping strategies based on an OpenCV computer vision library. The framework spans different levels of abstraction, performing high-level analysis using SystemC/TLM hardware models as well as validation of actual RTL models on a FPGA target platform.

Other high-level exploration approaches target specific design goals instead of concentrating on functional verification. Zuo et al. [4] present a framework generating SystemC models for exploration purposes towards low-power architectures. Based on a subset of C/C++ (specifically affine program regions) the framework computes power and performance characteristics to subsequently generate realistic SystemC models of a corresponding accelerator component. The generic nature of the generated architecture deliberately targets early design stages.

Mück et al. [5] presents an approach to integrate hardware and software components into a virtual platform based co-simulation environment. It is demonstrated how to utilize metaprogramming techniques common to OOP to facilitate transparent communication between partitions of functionality implemented in hardware or software. In addition, a NoC interconnect is provided to facilitate a higher number of actively communicating hardware accelerators.

In Grüttner et al. [6] a rapid prototyping framework for heterogeneous SoC is presented. It leverages high simulation performance by executing target software natively on the host system. Comparable to the simulation of hardware modules, the software is annotated with power and timing information to achieve realistic exploration results. Here the designer must evaluate the trade-off between performance and accuracy.

Typically, these DSE frameworks aim to identify the most suitable architecture without exposing the complexity and effort of actual hardware development to the user. This became possible by the growing number of high-level synthesis tools, generating RTL hardware models based on available (high-level) software code.

A recent survey of different commercial and academic high-level synthesis tools can be found in Nane et al. [7] These tools can be specifically targeted to a domain as well as general purpose solutions. Addressing the advantages and disadvantages of different approaches of HLS is beyond the purpose of this work and following only a few tools are exemplary presented.

One example is CatapultC [8] (product of Calypso Design Systems; used in Mück et. al.), it generates synthesizable cycle accurate code based on C/C++ code or already available SystemC models.

Other solutions also provide functionality beyond synthesis; Legup [9] demonstrates how profiling information can be directly utilized to decide on a hardware/software partitioning (i.e. offloading compute intensive software sections to a FPGA fabric). By skipping the DSE step on a higher abstraction level this either requires time-consuming simulation runs on RTL or the availability of the target hardware.

To address high-level DSE, the generated RTL designs themselves are typically not directly utilized as their simulation performance is not sufficient. However, they are a good source of information regarding the expected performance and power consumption of the actual hardware accelerator. Relaying those parameters back into the high-level models allows the simulation environment to deliver accurate estimations used in the DSE.

To the best of our knowledge, the solution presented in this work is unique as it integrates seamlessly into a state of the art SystemC/TLM2.0 virtual platform without exposing all its complexity to the designer exploring different hardware/software partitioning choices.

III. High-level Exploration of HW-SW Partitioning Scenarios

Diminishing improvements in power consumption and computational performance through technology scaling forces system engineers to deviate from homogenous architectures of the past. More and more we transition into a heterogeneous age of computing [10]. Gains in performance and reduction in power consumption now typically stem from specialized hardware accelerators tailored for specific workloads. Thereby tightening the interdependence between hardware and software, breaking with the traditionally sequential development of each. For the system designers' point of view, this dramatically increases the responsibility to choose the right partitioning between the two domains. Where in the past the designers experience was sufficient, today's increased design space requires new methodologies to reliably select the most suitable architecture.

One established trend to tackle the continuously growing design space is composed of two components: fast hardware/software co-simulation (on a high abstraction level) and intelligent strategies to parse the design space to reduce the number of necessary simulation runs. This work aims to improve the first component; specifically, in supporting the designer to evaluate different hardware/software partitioning in a quick and easy fashion. Works to reduce the number of necessary simulation runs to properly evaluate a designs space towards an application can be found in [11-16].

When analysing different hardware/software partitioning scenarios, it is appealing to directly compare the raw computation performance of the general-purpose processor with the specialized accelerator. However, this is a very skewed view on the world of co-design and its pitfalls. To accurately estimate the achievable performance, one must observe the complete interaction between accelerator, memory and software system in its entirety [17]. This holds true for the analysis of both the performance and power/energy consumption.

New and upcoming architectures embrace the use of specialized accelerators or co-processors, by providing cache coherent access to the main-memory from different heterogeneous compute resources. One example for this is the Coherent Accelerator Processor Interface (CAPI) developed by IBM [18] or the Heterogeneous System Architecture (HSA) [19]. While available, it is not commonly used especially in the low power domain, as achieving cache coherency for multiple heterogeneous actors is costly in chip area and power consumption. These coherent architectures are not within the scope of this work. We investigated the classic approach, where accelerators have their own fast memory without direct access to the main-memory. The required data movement causes runtime overhead, reducing the achievable performance and increasing the power consumption. Eq. (1) illustrates the individual components of the timing overhead incurred by the interaction between the accelerator, memory and software system. Here $t_{receive}$ and $t_{transmit}$ account for the time needed to move data to and from the accelerator. Additional time is needed to decode and encode the serialized data stream to be used by the computation and preparation for the data to be moved back to the main-memory (t_{decode} and t_{encode}). The actual computation time required by the accelerator is represented by $t_{computation}$.

$$t_{acc\_total} = t_{receive} + t_{decode} + t_{computation} + t_{encode} + t_{transmit} \tag{1}$$

Data handling can be achieved through different means; in the remainder of the paper we will look at a software and a DMA controller based method and their impact to the overall computation time.

While the achievable performance is the central optimization goal, it cannot be viewed in isolation. A system's power and the resulting energy consumption are becoming increasingly important. This work includes early power estimation models based on concepts proposed in [20]. Power consumption is composed of two main components: static and dynamic. Static power is based on the cell leakage current and is directly proportion to silicon area. Dynamic power is the sum of internal and switching power and is linearly dependent on the clock frequency. These parameters are annotated to the accelerator hardware model and contribute to the overall power consumption estimated by the virtual platform used in SW2TLM.

IV. Design Flow

This section will illustrate the design space used for the architectural exploration, the individual tools comprising the automated framework and concluding with a look at the user interaction and usability of the framework.

A. System and Accelerator Architecture

The baseline architecture used for the design space exploration is shown in Figure 1. It contains all necessary components to accurately reproduce the behaviour of a complete SoC. Software is executed by a LEON3 instruction set simulator (Generated using TRAP[21]) that is embedded into an accurate MMU/cache architecture. Communication is handled via two busses (high-performance and peripheral) connecting the remaining components. The investigated accelerators are shown top right, they can be instantiated as AHB-Masters as well as slaves depending on the data transportation method.

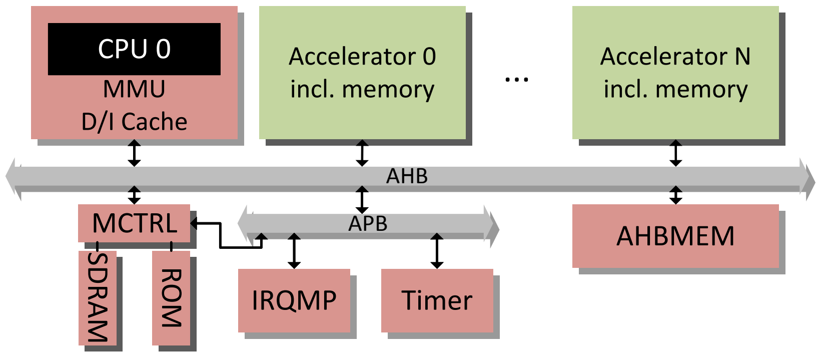


Figure 1. Baseline SoC architecture based on SoCRocket IPs

As described before, the explored architectures are based on fast local memories included in the accelerators itself. Consequentially input and output data must be moved to and from the accelerator. Here we differentiate two options: first data is handled directly from the software and second data is handled by a DMA controller within the accelerator to move the data independently. The structure of the accelerator and the control flow is illustrated in Fig 2. When data handling is implemented directly in software, the accelerator is used as an AHB-Slave with a simple blocking function call to perform the calculations. Using this technique, there is no need for additional synchronization between ISS and accelerator as the execution is blocked until the output data becomes ready. While the TLM implementation is straight forward, handling large data sets in software causes significant overhead compared to moving the data directly from the memory to the accelerator. This is implemented in the second option. For this the accelerator is extended to include DMA functionality, independently reading and writing data to and from the memory. This reduces the CPU load and can leverage efficient bus bursts. To perform memory accesses on its own, the accelerator now is integrated as a AHB-Master and Slave. Here the software is only required to provide the addresses and length of the data to be computed. On the other hand, synchronization now must be implemented explicitly. As the architecture includes an interrupt controller (IRQMP) it can be used to signal the end of computation and data movement. The difference in communication overhead and impact on the overall performance is shown and discussed in section V.

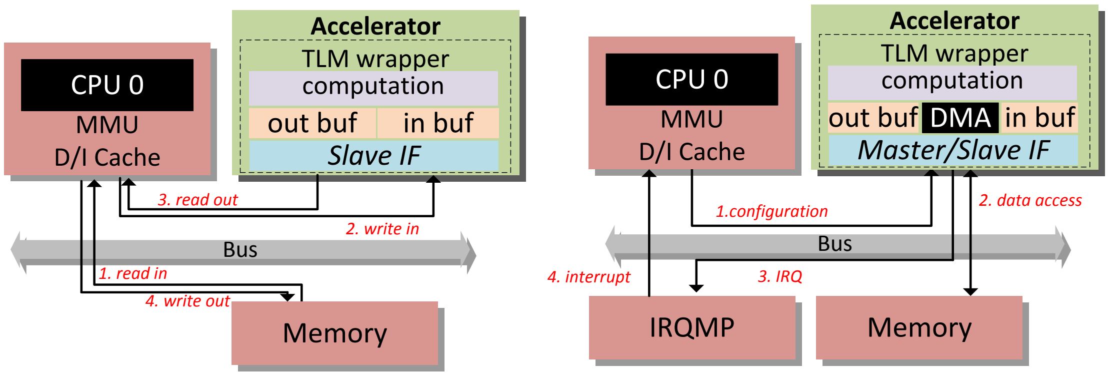


Figure. 2 Accelerator architecture and data flow (SW left, DMA right)

B. *Tool Flow*

Hiding complexity of the hardware/software co-design process from the user is a central goal of the proposed SW2TLM automation framework. Its structure and contact points with the user are shown in Figure 3. Based on the general architecture defined above, the user only needs to supply a characterization of the software section to be accelerated. Using this input, the framework isolates the specified software functionality and encapsulates into a TLM wrapper. This wrapper provides the functionality needed to communicate with the accelerator over the bus and as well as signal the processor when the computation is completed.

For the software functionality to be useful as part of a suitable hardware model, it needs to be annotated with a realistic (or at least pessimistic estimations of) computational delay and power consumption figures of corresponding hardware implementation. These parameters are generated using a state of the art HLS and RTL synthesis tools (like Vivado HLS and Synopsys design compiler). Instead of directly interfacing the generated RTL hardware model into the virtual platform, only the required information is extracted from the corresponding synthesis reports. Using the RTL model directly in the virtual platform would severely degrade the simulation performance and require a complex transactor implementation to bridge clocked (RTL) and un-clocked (TLM) sections of the investigated architecture.

In the next stage, the framework integrated the accelerator into the existing software control flow. To this end, a software wrapper is constructed to facilitate the communication and synchronization with the accelerator hardware. For the computation, it is necessary to serialize the possibly complex data structures that need to be written into the input buffer of the accelerator (see Figure 3). The same applies for the results of the computation stored in the output buffer of the accelerator.

Subsequently the virtual platform is configured to instantiate the generated accelerator transaction level (TL) model and load the modified software. The resulting executable simulation model is then used to capture the previously described performance and power parameters. After the simulation, these are summarized in a compact and humanly readable report file.

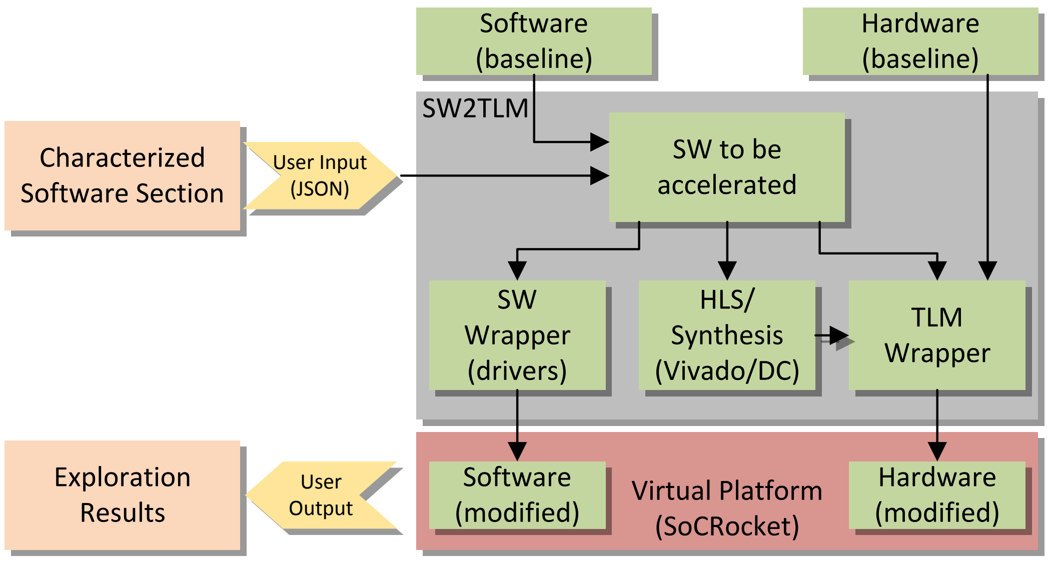
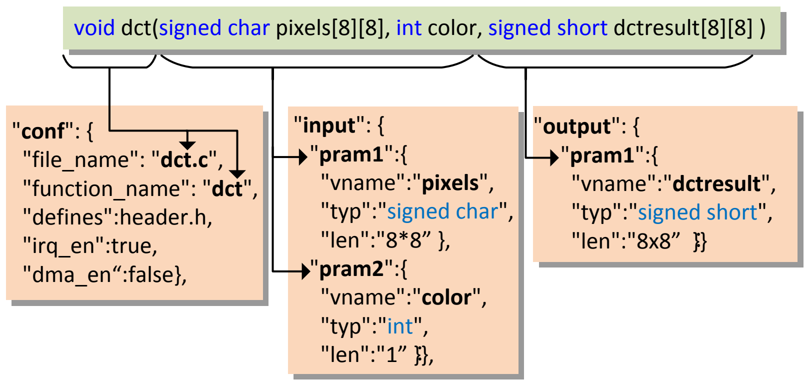


Figure 3. SW2TLM automated tool chain and user interaction

C. *Usability*

To further illustrate the high usability of the framework, Fig. 4 shows an exemplary software characterization to be provided by the user. The content of the JSON file (content of the three lower boxes) indicates the location of the source file storing the software function to be accelerated as well as a list of inputs definitions and the means of synchronization. For example, the field "irq_en" indicates that the processor is notified via interrupt after the accelerator has finished the computation. With the Field "dma_en", the user can select the memory management applied for the accelerator. Enabling the DMA will include a dedicated controller in the accelerator, to read and write from the main-memory independently. If disabled, the software wrapper is extended with additional code to transport input and output data to and from the accelerator (see Fig. 2). The differentiation of variables into inputs and outputs and the annotation of type and length are used to generate decode and encode functions needed to serialize and de-serialize the data.



```
void dct(signed char pixels[8][8], int color, signed short dctresult[8][8] )
```

```
"conf": {
  "file_name": "dct.c",
  "function_name": "dct",
  "defines":header.h,
  "irq_en":true,
  "dma_en“:false},
```

```
"input": {
"pram1":{
  "vname":"pixels",
  "typ":"signed char",
  "len":"8*8” },
"pram2":{
  "vname":"color",
  "typ":"int",
  "len":"1” }},
```

```
"output": {
"pram1":{
  "vname":"dctresult",
  "typ":"signed short",
  "len":"8x8”  }}
```

Figure 4. JSON based characterization of a software function

V. Evaluation

Image processing applications are often complex and require high performance, which make them suitable candidates for hardware/software co-design. The extensive computation of an application such as feature extraction achieves higher performance and efficiency in hardware while the highly complex tasks like parameter classification are implemented in software. In our experiments, we selected three applications: JPEG Codec, image filtering and image integration to evaluate the SW2TLM framework.

JPEG Codec: is widely used lossy image compression format. The main parts of JPEG encoder are: Color Space Conversion, Chromatic Sub-Sampling, Block Splitting, Discrete Cosine Transformation (DCT), Quantization and Entropy encoding. We choose the **DCT** functionality for our hardware/software exploration and automatically generated a corresponding hardware accelerator while the remaining functionality stays in software. **Image Filtering:** is a sample application that implements a moving average filter which is a low pass FIR filter commonly used for reducing the noise in an image. The filter kernel is modelled as a hardware accelerator (**IF**). **Image integration:** is used in computer vision algorithms e.g. speed up robust features (SURF) and scale invariant feature transform (SIFT) form image pyramid implementation. Here, the image integration algorithm is ported into a hardware accelerator (**II**).

A. *Performance and Power Parameters*

Table 1 summarizes the timing parameters used for the performance evaluation. Most of the parameters were already defined in Section III. Further parameters are: Total communication overhead introduced by the accelerator ($t_{com-overhead}$), execution time of the characterized software section running on the ISS (t_{sw}) and the absolute application runtime independent of the chosen implementation (t_{app}).

Table 1 Timing parameters evaluated in the performance analysis

Parameter Name	**Description**
$\boldsymbol{t_{receive}}$	Writing data to HW ACC directly or via DMA
$\boldsymbol{t_{transmit}}$	Reading data from HW ACC directly or via DMA
$\boldsymbol{t_{decode}}$	Decoded data in ACC
$\boldsymbol{t_{encode}}$	Encode data in ACC
$\boldsymbol{t_{computation}}$	HW ACC computation time
$\boldsymbol{t_{com-overhead}}$	Communication overhead
$\boldsymbol{t_{acc\_total}}$	Total HW ACC execution time
$\boldsymbol{t_{sw}}$	Execution time in of selected software section
$\boldsymbol{t_{app}}$	Absolute application execution time

The investigated Parameters regarding the power consumption are listed in Table 2. Static ($p_{leakage}$) plus dynamic ($p_{internal} + p_{switching}$) is used to calculate the total power consumption (p_{total} Eq. (2)) of a specific hardware component as well as the complete platform. These parameters are represented as the average power draw, in addition the consumed energy until task completion is calculated ($\boldsymbol{e_{total}}$ Eq. (3)).

$$P_{total} = P_{leakage} + P_{internal} + P_{switching} \tag{2}$$

$$e_{total} = P_{total} * t_{app} \tag{3}$$

Table 2 List of investigated power and energy consumption parameters

Parameter Name	**Description**
$\boldsymbol{p_{leakage}}$	Static power consumption
$\boldsymbol{p_{internal}}$	Internal part of dynamic power consumption
$\boldsymbol{p_{switching}}$	Switching part of dynamic power consumption
$\boldsymbol{p_{total}}$	Total power consumption
$\boldsymbol{e_{total}}$	Total energy consumption

B. Performance Analysis

To demonstrate the functionality of the SW2TLM automation framework, it is applied to three different applications identified in the beginning of this section. For each we compare the performance of the accelerator with and without DMA against the performance of the pure software solution. To illustrate the difference in ratio of communication (see Fig. 5) overhead vs. actual computational delay, the total computation time is broken down into the individual components shown in Table 1. In summary, the main findings are:

- **Decode/Encode:** It is directly visible, that t_{decode} and t_{decode} contribute an insignificant delay to the overall accelerator delay. This effect will be more prominent for the complex data types; the structure of the tested image formats is mostly sequential in nature and requires very little encoding and decoding effort. Of the tested algorithms, the only visible encoding effort is observed for the DCT, here data type conversion is necessary to account for the difference in endianness between hardware accelerator and the emulated LEON3 core.
- **Execution time (accelerated section only):** our tests show the image integration got less speedup (5.46x with accelerator and 8.08x including the DMA), this is mainly due to the simplicity of the algorithm's operations (only pixel summation). The DCT and image filtering contain complex operations, which are better candidates for hardware acceleration, DCT is 20.8x and image filtering is 28x faster using the accelerator (with DMA controller 32.3x and 35.9x) than the software implementation.

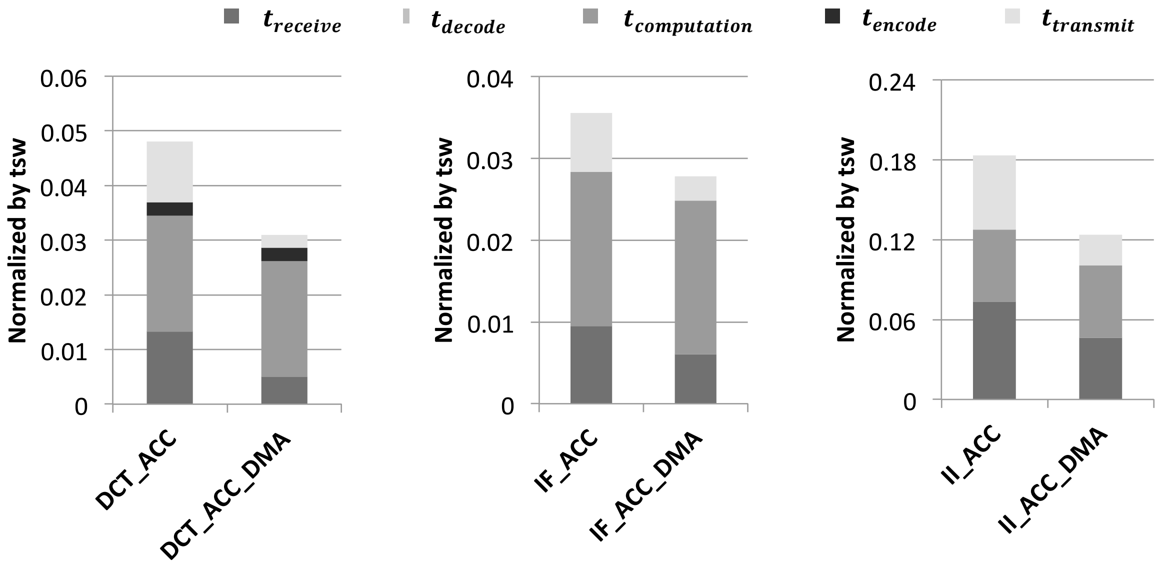


Figure 5. Combined accelerator delays; DCT left, Image Filter middle, Image Integration right

- **Absolute execution time**: Fig. 6 shows the speedup factors achieved over the complete application execution time in software. The highest speedup is achieved for the image filtering (up to 6.5x). The difference between software and DMA data handling accounts for up to 0.3x absolute speedup.

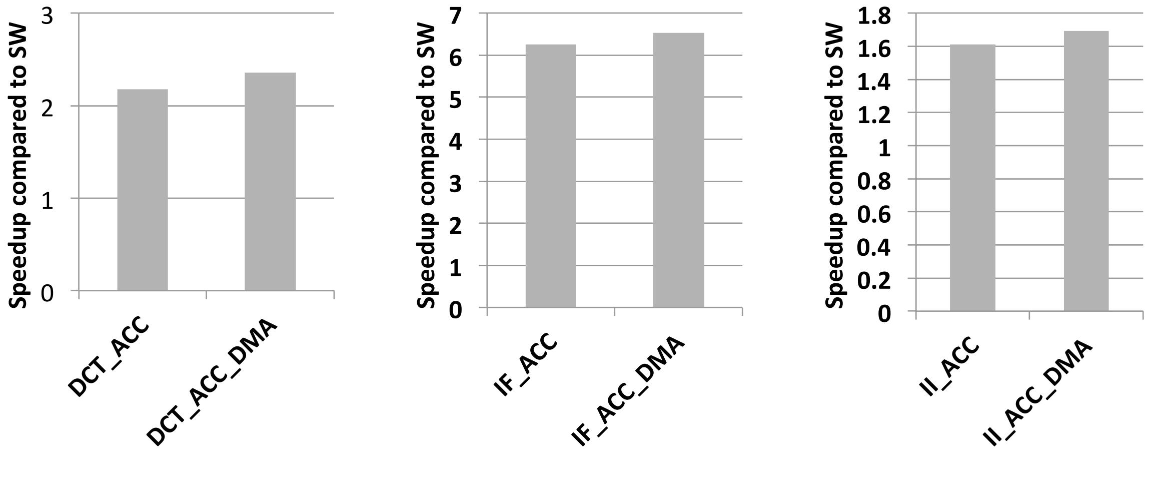


Figure 6. Absolut application speed up; DCT left, Image Filter middle, Image Integration right

- **Communication overhead**: calculated as shown in Eq. (4):

$$Co_t = \frac{t_{receive} + t_{decode} + t_{encode} + t_{transmit}}{t_{acc\_total}} \tag{4}$$

Fig. 7 shows the communication overhead ratio recorded in the simulation. As expected, the highest communication overhead was measured for the software based data handling (ranging from 47% up to 70%). When data is moved using the DMA controller a reduction of at least 14% was observed.

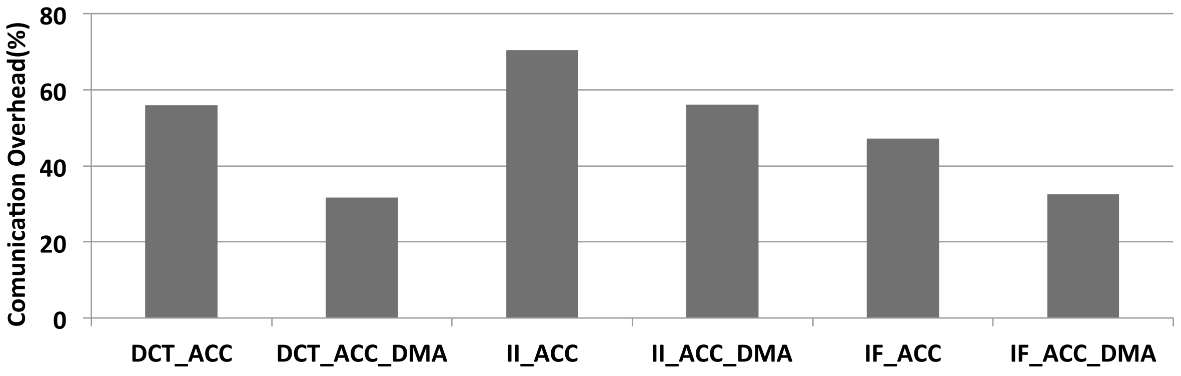


Figure 7. Communication overhead shown in percentage of complete accelerator delay

- **Power and Energy:** The power model applied to the ISS does not incorporate typical power saving techniques when idle. Modelling these techniques on the system-level is still a topic of research. Therefore, the average power draw increases (see Fig. 8) when the accelerator hardware is added to the system. This increase ranges from 0.2 % up to 19%. The high values of the image filtering are caused by frequent data accesses related to the size of the filter kernel. Despite this limitation, the overall energy consumption exhibits a significant improvement. The observed energy consumption ranges between 18% and 62% of the original software implementation.

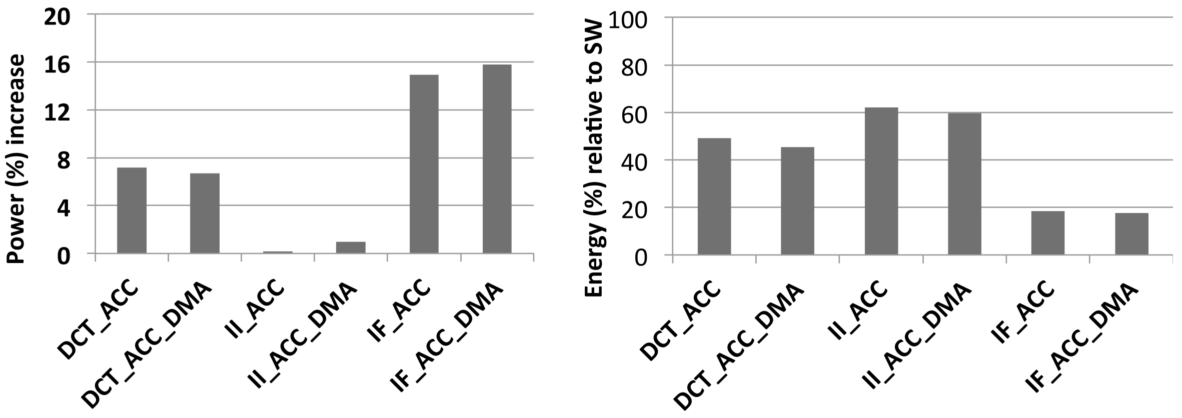


Figure 8. Power consumption increase to SW (left) Energy consumption relative to SW (right)

VI. Conclusion and Future Work

In this paper, the automation framework SW2TLM is demonstrated using real world image processing applications. It supports engineers in the increasingly difficult design space exploration of complex hardware/software systems. Users are only required to characterize the investigated software section to generate a corresponding accelerator model embedded into a complete virtual prototype on the TL-level. The framework leverages a sophisticated virtual platform to explore the accelerators behaviour and the overhead incurred from the communication. The reported communication, performance and power parameters are a valuable foundation to base future architecture decisions on. The presented evaluation illustrated how important it is to include communication overhead into the exploration space, as it can account for most the overall accelerator computation time (here up to 70%). Also, capturing accurate overall energy consumption is essential for the design in constrained or battery dependant domains.

Upcoming development on the automation framework will address the following points: first, the estimation of the accelerator power consumption will be improved by investigating different implementations of memory cells through the synthesis tool as well as considering zero-copy accelerator architectures. Secondly, extending the support of complex data structures of the automatic decoder and encoder function generation. This issue is known in the HLS domain, as data handling and organization in software observes a far greater degree of freedom compared to hardware implementation. And lastly, extending the virtual platform by modern low power techniques to investigate the effects of dynamic frequency and voltage scaling when offloading computational tasks to dedicated accelerators.

References

[1] T. Schuster, R. Meyer, R. Buchty, L. Fossati, and M. Berekovic, " SoCRocket - A Virtual Platform for the European Space Agency's SoC development" in Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on. IEEE, 2014.

[2] R. Meyer, J. Wagner, B. Farkas, S. Horsinka, P. Siegl, R. Buchty, and M. Berekovic, "A Scriptable Standard-Compliant Reporting and Logging Framework for SystemC" in ACM Transactions on Embedded Computing Systems (TECS), 2016.

[3] M. Mefenza, F. Yonga, L. B. Saldanha, C. Bobda and S. Velipassalar, "A framework for rapid prototyping of embedded vision applications" in Design and Architectures for Signal and Image Processing (DASIP), 2014 Conference on, Madrid, 2014.

[4] W. Zuo, W. Kemmerer, J. Bin Lim, L. Pouchet, A. Ayupov, T. Kim, K. Han, and D. Chen, "A Polyhedral-based SystemC Modeling and Generation Framework for Effective Low-power Design Space Exploration" ICCAD, 2015.

[5] T. R. Mück and A. A. Fröhlich, "Seamless integration of HW/SW components in a HLS-based SoC design environment," International Symposium on Rapid System Prototyping (RSP)*,* Montreal, QC, 2013.

[6] K. Grüttner, P. A. Hartmann, T. Fandrey, K. Hylla, D. Lorenz, S. Stattelmann, B. Sander, O. Bringmann, W. Nebel, and W. Rosenstiel "An ESL timing & power estimation and simulation framework for heterogeneous SoCs," Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2014.

[7] R. Nane, V. M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools" in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015

[8] Mentor Graphics. "Catapult C synthesis." *Website: http://www. mentor. com* (2008).

[9] B. Fort, A. Canis, J. Choi, N. Calagar, R. Lian, S. Hadjis, Y. T. Chen, M. Hall, B. Syrowik, T. Czajkowski, S. Brown, and J. Anderson, "Automating the design of processor/accelerator embedded systems with legup high-level synthesis", in IEEE Embedded and Ubiquitous Computing (EUC), 2014 .

[10] J. Teich, "Hardware/Software Codesign: The Past, the Present, and Predicting the Future," in *Proceedings of the IEEE*, 2012.

[11] S. A. A. Shah, B. Farkas, R. Meyer, and M. Berekovic, "Accelerating MPSoC Design Space Exploration Within System-Level Frameworks" in IEEE Nordic Circuits and Systems Conference (NORCAS), Copenhagen, Denmark, 2016.

[12] F. Angiolini, J. Ceng, R. Leupers, F. Ferrari, C. Ferri, and L. Benini, "An integrated open framework for heterogeneous MPSoC design space exploration," in DATE, 2006.

[13] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," in Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems, 2002.

[14] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, “Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications,” in Embedded Computer Systems (SAMOS), 2011.
[15] F. Herrera and I. Sander, “Combining analytical and simulation-based design space exploration for time-critical systems,” in Specification Design Languages (FDL), 2013.
[16] Z. J. Jia, A. N´u˜nez, T. Bautista, and A. D. Pimentel, “A two-phase design space exploration strategy for system-level real-time application mapping onto MPSoC,” Microprocessors. Microsystems, 2014.
[17] Y. S. Shao, S. Xi, V. Srinivasan, G. Y. Wei, and D. Brooks, "Co-designing accelerators and SoC interfaces using gem5-aladdin," in international Symposium on Microarchitecture (MICRO). 2016.
[18] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel, " CAPI: A coherent accelerator processor interface" in IBM Journal of Research and Development 59.1, 2015.
[19] Kyriazis, George. "Heterogeneous system architecture: A technical review," in AMD Fusion Developer Summit, 2012.
[20] S. A. A. Shah, J. Wagner, T. Schuster, and M. Berekovic, “A lightweight- system-level power and area estimation methodology for application specific instruction set processors,” in PATMOS, 24th International Workshop, 2014.
[21] Fossati, Luca. "IP-SOC 2010."