# SEAMLESS HW/SW CO-DESIGN FLOW

Jože Dedič, Andrej Trost, Andrej Žemva

Faculty of Electrical Engineering, University of Ljubljana, Slovenia

**Key words:** design space exploration, directed acyclic hyper-graph (DAG), HW/SW co-design, partitioning, scheduling

**Abstract:** Computing applications complexity has raised to the level where managing the design flow in the classical way, while satisfying various constraints, is becoming extremely hard to cope with. We see two main reasons for that. The first reason is partial consequence of Von Neumann architecture inheritance which imposes throughput restrictions /1/ with its imposed program sequentializing. The CS curriculum and rich set of tools are both suited to that model. Now, speedups are possible by providing additional HW components operating concurrently. We expect that the work will be done in the direction of a revised application development design flow approach which would trade the execution time for complexity. The second reason is the consequence of the IC manufacturing technology improvement with its increasing level of integration which enables a steep system level complexity increase in a wide range of applications. Many of them also face short time to market. CAD tools are not in pace with this increasing complexity, thus putting pressure onto design teams. The design cycle round time shortening is possible by different levels of modeling, where each of them features design decision estimations. This paper presents the HW/SW co-design architecture exploration space and gives an overview over the related methodologies. Based on the study of these methodologies and our experience with an ad-hoc approach, we present a seamless HW/SW co-design flow. The flow forms the basis for the development of a CAD tool helping designers to considerably benefit from the HW concurrency and offering an efficient system level approach.

# Enovit načrtovalski potek sočasnega načrtovanja strojne in programske opreme

**Kjučne besede:** hkratno načrtovanje strojne in programske opreme, razmeščanje, razvrščanje, usmerjen graf vezja

**Izvleček:** Kompleksnost računalniško podprtih aplikacij je narasla do nivoja, ko je klasičen potek načrtovanja, ob hkratnem upoštevanju vseh omejitev, postal zelo težko obvladljiv. Tukaj vidimo dva glavna razloga. Prvi razlog je delna posledica Von Neumannove arhitekturne zapuščine, ki z vpeljano programsko sekvenčnostjo omejuje podatkovno pretočnost /1/. Računalniška veda in nabor orodij sta oba prilagojena temu modelu. Pohitritve so sedaj mogoče z dodajanjem vzporedno delujočih strojnih komponent. Nadaljnje delo vidimo v smeri predrugačenega načrtovalskega poteka razvijanja aplikacije, ki bi ceno povečane hitrosti izvajanja aplikacije plačal s povečanjem arhitekturne kompleksnosti. Drugi razlog je posledica tehnološke izboljšave izdelave integriranih vezij, ki z vse večjo integracijo omogoča eksponentno naraščanje sistemske kompleksnosti na širokem področju aplikacij. Dodaten pritisk pri razvoju aplikacije pa je lahko tudi kratek čas do trga. CAD razvojna orodja pa ne uspejo držati koraka v kompleksnosti, tako se povečujoče se breme prenaša na raziskovalno ekipo. Skrajšanje načrtovalskega razvojnega cikla je mogoče z modeliranjem na različnih nivojih, ki podpirajo možnost ocenitve načrtovalskih odločitev. Članek predstavi arhitekturni nabor primeren za hkratno načrtovanje strojne in programske opreme in poda pregled sorodnih metodologij. Na osnovi študije teh metodologij in izkušenj pridobljenih z ad-hoc pristopom predstavljamo enovit načrtovalski postopek sočasnega načrtovanja strojne in programske opreme. Načrtovalski potek vidimo kot osnovo za razvoj CAD orodja, ki bi načrtovalcem omogočil efektivno izkoristiti prednosti sočasnega delovanja strojne opreme in bi nudil efektivni sistemski pristop.

## 1 Introduction

Size and complexity of high performance signal, image and control processing algorithms is increasing tremendously. Classical SW approaches with traditional von Neumann-like architectures are far from being optimal. Their major strategy to overcome complexity and increase throughput is increasing the processor clock speed and SW optimization methods /1/, /2/. Moreover, the algorithm complexity and real-time constraints in reactive embedded systems can be so demanding that classical high end SW processor at a reasonable clock speed can no longer manage the task /3/. The only possible solution to the problem is a unified HW/SW co-design approach. Nevertheless, when studying HW/SW co-design, some other issues of similar importance arise. When partitioning between HW and SW, the following design metrics have to be accounted for: cost, size, performance, power, time-to-prototype, time-to-market, correctness, safety, and maintainability /5/. As some of these design metrics compete among themselves, managing the HW/SW co-design exploration space is even harder. In order to meet the optimization challenge, the designer must be comfortable with a vast variety of HW and SW implementation technologies enabling him/her to find an optimal solution for a given application and constraints. A rich expertise in both SW and HW domain is required for this purpose.

After the heterogeneity problem is solved, the next to be copped with is the system level complexity problem. The process of applying the application description onto architecture can be an extremely complex task when details are to be described in a non hierarchical way. Support for different level approaches has to be provided for. System level design decision choices should be available as late as possible within the design cycle, thus enabling the explore-propose-validate-refine process to achieve its best, while the accuracy of a model gradually increases. This is not the case in the classical approach where HW (processor) is designed in advance, hopefully powerful enough, and SW (application) is adapted to it. Each designer involved with designing a fairly modest system, from the initial specification to the final implementation, has to deal with complexity when interconnecting heterogeneous components. The explore-propose-validate-refine process can be a very tedious work, especially when a variety of possible implementations should be explored to best satisfy design metrics. In the first approach aiming at lowering the designers' stress while simultaneously increasing productivity and the largest manageable system complexity, the design work is partitioned among multiple designers. Two problems arise. First, when partitioning work on smaller subsets, the designers specialize into a relatively narrow segment of the whole system. This gives rise to the problem of how developing individual parts of the system without having a clear idea of the overall system integrity. Second, adding system designers to the project does not work as expected. Believing that the designer's productivity is independent of the project team size is not in place. Simple man-month relations are not valid when the project team size increases. After some point, enlarging the number of designers working on the project does not contribute to the design cycle shortening/5/.

All these facts show the importance of research studies for guiding development of CAD tools to support the entire design flow based on heterogeneous architectures and to provide system level support. Such CAD tools are necessary to cope with the exponentially increasing system complexity. Already existing CAD tools provide a semi-automatic interactive environment where most important scheduling and partitioning decisions are the designer's choice. As seen currently, a fully automatic approach is impossible for the present.

The rest of the paper is organized as follows. Section 2 describes diverse architectures enabling efficient HW/SW co-design exploration. Analyzed are also their weaknesses and strengths. Section 3 presents some of the work related to HW/SW co-design approaches and methodologies. In section 4 we propose a seamless HW/SW design flow based on the study of these methodologies and our experience with ad-hoc approaches of application partitioning and system level integration /15/. In section 5 we introduce our ad-hoc application partitioning with which we acquired knowledge needed for mastering heterogeneous architectures and HW/SW partitioning. Section 6 concludes this paper.

## 2 HW/SW co-design exploration opportunities

HW/SW design deals with balancing the architecture resources of a digital system in the search for an optimal implementation. With the term architectural resources we denote all kinds of storage resources (memory, registers), programmable resources (FPGAs), partially programmable resources (application specific and general purpose SW processors), nonprogrammable resources (single purpose processors) and communication resources (interconnections, buses) which provide space for flexible exploration. All resources should be taken into account and trimmed carefully to achieve optimal design metrics. Design metrics that we are focusing on in our work are cost, time-to-prototype and performance improvement. Meeting these criteria leads us to some compromises that still permit us a level of flexibility large enough to explore a variety of implementation options. Some possible exploration environments that enable HW/SW co-design study are:

- FPGA-only HW environment with a processor implemented as a softcore. Microprocessor architectures implemented as a softcore (ARM, ARC, MIPS, PowerPC, etc.) offer limited performance. There are many technical reasons for this. Far ahead in usage are Xilinx's Micro Blaze (for the Spartan and Virtex family of FPGAs) /12/ and Altera's Nios (for Cyclone and Stratix FPGAs) /13/ sold as IP cores. These processors can achieve the maximum clock speed in the range of 100MHz and can occupy quite a large portion of FPGA; the more complex the processor is the slower the speed is. OSs can be very simple and provide only some basic functionality thus making the portability of application harder. Advanced OSs require a more powerful processor, more FPGA resources, but also decrease the maximal clock speed.
- Standalone processor with FPGA logic as its periphery /14/. Embedded systems are a very good example of it. Their maximal performance is attained if FPGA is connected directly to the processor's bus to optimize data transfer rates. A lower overhead for accessing external HW provides more partitioning possibilities. Despite their greater complexity and an additional HW overhead, an extensive computation power can be attained. The various OSs are well supported for many processors on the market with compilers, cross-compilers and debuggers. Many of them can be obtained free of charge. The initial cost includes building such an embedded platform and porting some OS to make the platform alive and stable.
- The PC environment and some additional custom HW /15/. PC does not represent any overhead as a result of its popularity and availability. Executing SW is stable and a variety of development tools exist. If communication method is not of primary interest (PCI bus), it offers a great set of architectural resources when combined with a resource-wealthy add in PCI board. Also, when moving to another set of architectural resources, it allows for a great amount of code reusability (SW or HW IP). Some HW scalability is also supported in the PC environment through PCI extension slots.
- New extensions to HW/SW co-design are offered by programmable SoC platforms. They comprise programmable arrays, hard wired microprocessors and rich set of fast communication peripheries. Their rep-

resentatives are Xilinx Virtex II Pro /12/ and Altera Excalibur /13/. Virtex II Pro consists of up to four PowerPCs. They are integrated within a regular FPGA structure by sacrificing some silicon that would otherwise be used for CLBs and interconnections. Excalibur offers an ARM9 processor and programmable array integrated within the same IC, with a smaller level of integration than the former. In this way, HW/SW cohesion can be applied very efficiently and the whole design can be finally fitted in one chip.

As a result of the overall complexity of the whole system design, regardless of the environment chosen, stable and efficient SW tools are required to efficiently manage application mapping onto available architectural resources.

## 3 Related work

There is currently a lot of activities related to HW/SW co-design methodology underway. Many of the research groups focus on some particular stages in the design process or even optimize some of these stages to best suit their finite extent of supported architectures. Such closed areas of increased interest can be the system level specification and modeling, partitioning and scheduling, compilation and synthesis, co-verification and co-simulation, automatic code generation for HW and SW interfacing, and automatic code generation for the task manager. The common denominator among them is splitting the input description into subtasks and describing data dependency between them. For that purpose most of them use some form of directed acyclic hyper-graph (DAG) /3/, /6/. Nodes in the graph represent subtasks (more or less complex operations) and edges represent data dependency between them. Many automatic scheduling and partitioning tools uses DAG (or some extension of it) for applying some optimization methods to obtain satisfactory mapping of applications onto architecture. The task of optimization methods is to find an optimal partitioning and scheduling scenario for subtasks extracted from the input specification. Common partitioning and scheduling problems belong to the class of NP-hard and intractable problems. Research studies have been done on algorithms involving heuristic search /7/. Heuristic optimization methods are guided by applied cost functions to evaluate implementation space realizations.

Numerous researches have already been done in the field of HW/SW co-design. Here we report only the work that we find particularly interesting for our study.

Wiangtong, Cheung, and Luk /6/ presented a semi-automatic co-design environment for a system consisting of a single general purpose processor and multiple reconfigurable HW units. Their study involves building HW/SW architecture suited for dataflow dominated applications. The proposed design flow enables input application description in high level language (HLL). Mapping the input description into DAG is done manually. Authors implemented automatic generation of the underlying code and taking care of necessary application sub-tasks communication by wrapping tasks in standard frames. Independent tasks are executed on several processing elements. They are controlled by an automatically generated task manager program running on the SW processor. Because of the task's standard frame overhead, this method is appropriate for coarse grain partitioning. Authors presented a study of heuristic methods suited for partitioning and scheduling /7/. They applied them onto DAG and thus made a next step towards a fully automatic design flow.

AAA methodology /3/ extends DAG and adds ability to specify loops through factorization nodes. This leads to an algorithm model called factorized data dependence graph (FDDG). Graph factorization consists of replacing a repeated pattern by only one instance of it. Because of extension, it is suitable both for data and control flow dominated applications. FDDG may be specified directly or it may be generated from HLL (Esterel, Signal). Methodology main efforts are towards graph transformations. Optimization consists of finding defactorization transformations within implementation space. This gives best results in terms of cost function (heuristics guided by their cost function). AAA uses a single factorized graph model from the algorithm specification down to the architecture implementation through optimizations expressed in terms of defactorization transformations applied to the algorithmic graph. Automatic generation of a HW implementation from an algorithm specification based on FDDG is supported by employing a set of rules for data and control path. The algorithm employs synchronization rules and a delocalized control approach (as opposed to the above mentioned methodology). Support for real-time extension is studied, too.

A very important area of HW/SW co-design is task communication in terms of resource sharing, which often does not get enough attention compared to its influence on the overall system performance. Communication channel is a resource, similarly as other processing elements, and must be scheduled. When several tasks use the same communication resource, the channel activity also causes task delays which must be taken into account when task scheduling. /9/ gives an overview of this topic, proposes rules and explores genetic algorithm heuristics to schedule tasks. While achieving shorter execution time, implied rules impose only a small overhead to the whole scheduling and partitioning process.

Work has also been done in the direction of finding a language that would meet the needs for describing HW and SW so as to enable compilation and synthesis, and support various level application modeling. Several mature languages exist that were originally suited for SW (C/C++) or for HW (VHDL/ Verilog) design. They all exhibit some weak points when bridging the heterogeneity gap. Some special points of interest are: support for HW description, concurrency support, system level description and modeling, gradual model refinement, and verification. SystemC /10/ solves this problem by introducing specific class libraries which are ANSI C++ compliant. SystemC benefits from all C++ object oriented attributes and leverages it by introducing concurrency, notion of time and support for HW data types. Extensions are realized through running an executable system description under the SystemC simulation kernel. SystemC tends to become a standard as a language-based modeling tool for system-level design; OSCI has already submitted it to the IEEE for standard approval. SystemC itself should not be considered as a methodology. It is a modeling language from which HW/SW co-design can benefit. Another very important feature is system verification support. Support is enabled through Cadenece SystemC verification extension built on top of SystemC library /11/.

## 4 Seamless design flow

If we outline some properties, which in our opinion the designer-friendly and applicable HW/SW co-design CAD tool should have, we quickly find some weak points of methodologies in many of the currently active research studies in the field of HW/SW co-design. The CAD tool, which we are steaming to, should take the advantage of mature languages and just fill the gap caused by heterogeneity. Input description languages that are already widely accepted and have a rich set of underlying supporting tools should not be disregarded and, for the same reason, new description languages should not be forced by any means. A work around could be implementing some additional features to the already existent languages (by means of libraries or language extensions of a reasonable level) or building some supporting environment to extend the language description capability. Tools for building SW executives (compilers) are already well optimized, and designers are trained to use them efficiently. Tools for building HW netlists (synthesis tools) are also very powerful. In this paper we are introducing a seamless environment where these sets of already existent tools can be used in a uniform way to support design flow from the system level description to distributed executives and net-lists. Effort should be made in the direction of automatically crossing the HW/SW barrier and at the same time reusing powerful aspects of tools on both sides.

The gap between HW and SW is currently handled by the system designer, who is doing a tedious work of the explore-propose-validate-refine process. The model accuracy is gradually increased when more details are added. This consequently prolongs the time needed for the system model development and simulation. To reduce the time required for design space exploration evaluation of design choices should be supported earlier in the design process, which leads us to system level exploration.

Figure 1 outlines the traditional design flow. This approach is also known as Y-chart approach /16/. It introduces the main idea that seamless HW/SW co-design environment should provide for a sufficient support. The input specification consists of an architecture and application description as well as application constraints. HW/SW co-design approach main object is finding the best mapping of application onto available HW resources, while satisfying constraints. As the Y-chart suggests, the process of finding the optimal mapping consists of iterating cycles. The process of evaluating different possible solutions that are candidates to realize the application within given constraints is named design space exploration. The design space consists of a variety of spatial and temporal mappings and, as mentioned before, this problem can easy become unsolvable. Dashed arrows suggest the order of design space exploration. First, the design space built from a given architectural and application description is explored. If no solution within the design space satisfies constraints, the next step is to revise the application description in terms of algorithm speedups. The whole design cycle from the previous step is repeated. If widening the design space still does not produce satisfactory solutions, this is an indication that, within given architectural resources, application mapping cannot be made by realizing constraints. Another important aspect of the Y-chart is reusability since it enables mapping of multiple target applications onto candidate architectures in order to evaluate performance.

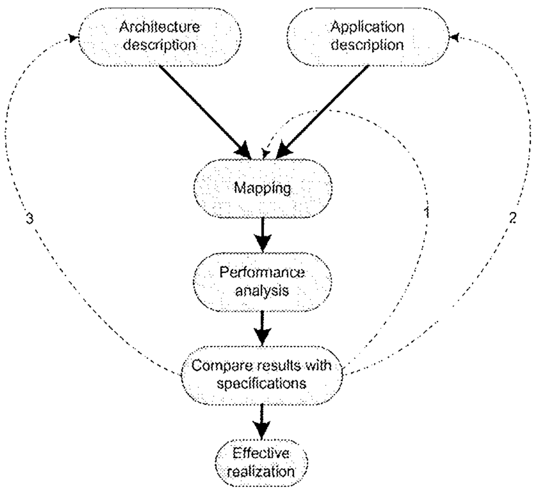

*Figure 1: Y-chart approach*

Our approach to HW/SW co-design partially follows the traditional Y-chart approach guidelines. It mostly extends it as it is in detail explained in Figure 2. The application is in the foreground. Optimal partitioning and scheduling are obtained by employing gradual model refinement.

The application description is split into the system level description and full-detail level description. At the system level description we benefit from SystemC system level modeling and model refinement. The system level application description satisfies two purposes. First, information about tasks and data dependency between them is captured to construct DAG. Second, the system level description is used as a simulation skeleton to guide heuristic methods so as to find the optimal spatial and temporal distribution. The task description can be very loose at this stage. The full-detail level is not needed at this stage and can be postponed.

Architecture is described by a set of available system resources, providing necessary external tools for synthesis, compilation, verification and simulation, providing IP blocks of more or less complex operators in terms of library, specifying standard frames to enable automatic generation of task communication and automatic generation of control logic for task scheduling, and specifying communication channels.

Similar to the application description, the architecture description is also split into two parts. The term architecture description means all kinds of specifications that smooth the higher level description compilation or synthesis onto target architecture (C/C++, VHDL, or an even higher level task representation abstraction). The coarse grain architecture description is provided with external tools that enable smooth transition from the HLL HW and SW code to the netlist (synthesis) and executable code (compilation) and provide its testing, simulating, verifying, debugging and profiling. Clearly defined architecture limitations that sensibly limit the design space size are also a part of the coarse grain description; e.g. a number of processing elements suitable for SW execution, amount of available memory, number of system buses... Library provides synthesizable and compilable description of standard elements, supports automatic generation of the underlying code and provides support for IP reuse. Library consists of blocks of HW and SW descriptions of various complexity levels. These can be all kinds of wrappers, supporting smoother integration of the user defined code (with HW or SW tasks), communication channels (implementing SW drivers and HW protocols), and various complexity level operators (from simple adders and multipliers to more complex cores such as DCT).

Although the split architecture description may look somehow artificially made, it is a necessary design approach, because programmable gate arrays enable realization of virtually any function, endlessly extending the design space. The coarse grain description is used for quick infallible partitioning and scheduling decisions, rejecting unfeasible schemes. Providing library of synthesizable cores wraps the endless design space to a final extension and enables IP reuse.

Constraints are used to build cost functions needed by heuristic methods to identify the best solution within the design space and to evaluate the result from heuristics. Constraints can be given in any combination of resource utilization, power consumption, and application execution time. Constraints must be later given appropriate weights to obtain the cost function to guide heuristic search methods.

Given the necessary input specification, the system level application description is studied and data dependency between tasks is obtained to build DAG. The main feature of DAG is determining the dataflow dependency to overcome the sequential nature of the application description and to discover parallelism possibility. Tightly connected to building DAG is rearranging parts of the graph by applying different algorithms, i.e. by increasing the parallelism rate and granularity modification /3/.

After application is split into subtasks and potential parallelism is discovered, DAG partitioning and mapping take place. Application is partitioned on the basis of the input specification about HW only, SW only and HW or SW tasks, and the design space, to be explored later, is defined. While partitioning and mapping, architectural resources information is needed to obtain a set of operators capable of executing application operations to be mapped. Up to this point the design space consists of a variety of combinations, covering every possible mapping of every subtask to appropriate available resource. In the case of implementation of HW resources with programmable circuits, the design space is infinite, thus practical limitation is set by a

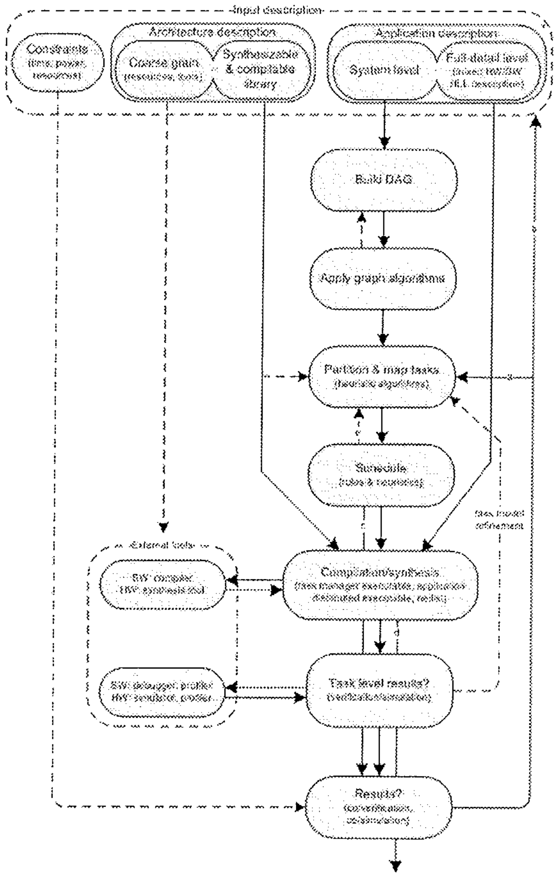

*Figure 2:* *HW/SW co-design design flow*

finite number of library components. Finding the optimal solution within the design space takes time that is exponentially dependant on the design space size when solved with feasible computers. Even in the case of a modest application, the problem quickly becomes unmanageable. Here, the system level approach enables us to explore only defined subsection(s) of the entire system, thus isolating the detailed level description of a partition and schedule enabled task from the rest of the system, described only for a necessary interaction. In the next step, scheduler extends the design space even more. Solving this kind of optimization problems belongs to the class of NP problems. Keeping the design flow time under control, heuristic methods are necessary. These methods will select a point from the design space and estimate its result adequation guided by the cost function. The way how the design space is explored depends on the chosen heuristic method /7/.

The partitioning step is tightly integrated with scheduling step heuristic methods and rules. The scheduler task is to find optimal temporal distribution which would produce the shortest application runtime. Temporal mapping is applied to subtasks that share common resources. The execution order depends on data dependency extracted from DAG and when determining the execution order, rules are applied which take into consideration resource conflicts and task delays caused by them /9/. Two resource allocation policies /4/ can be applied; dynamic and static. This paper addresses only the static one. To find optimal scheduling, each task must be described by its communication and processing time. This time can be obtained either as the input given approximately or as a more realistic feedback from subsequent stages. In Figure 2 it is depicted by dotted arrow labeled task model refinement.

After tasks are spatially and temporally mapped, the mapping efficiency can be estimated using SystemC system level model executive. It is indicated by a solid arrow labeled c. The input specification for SystemC executive is built from the system level application description input, modified by graph transformation algorithms and spatial and temporal mappings. Estimations are getting closer to realistic values when task descriptions are becoming gradually refined, which is the primary feature of SystemC. Partitioning and scheduling heuristic algorithms iteratively explore the design space until an optimal solution is found (depicted by a dotted arrow labeled a). If iterative heuristic algorithms fail to find a solution within the design space, the input description must be reviewed (depicted by a dotted arrow labeled b).

Gradual application system level model refinement introduces optimal spatial and temporal mapping for a given input specification. If results conform to constraints, the subtask descriptions should be refined to a full-detail level according to the winning partitioning scheme. Currently, we assume mixed SW and HW language description (C/C++, VHDL). At this point, architectural information is used to wrap detailed described subtasks into standard frames thus enabling automatic task connectivity and automatic control generation. Compilation and synthesis are done with the usage of external tools connected into a seamless design flow through command line extension. When constraints are satisfied, this is also the subsection of the design flow where the entire HW and SW code is generated for every programmable part of the architecture. After a synthesizable and compilable code is obtained for every task, it can be verified and simulated with the use of external tools. Task level verification and timing simulation can be applied using external tools, which will serve as an exact guide to the partitioning and scheduling algorithm.

## 5 JPEG design Example

As previously described, the design flow gradually evolved by taking into account the related work in this area, leveraged by our experiences obtained with an ad-hoc approach of partitioning and system level integration. Experiences with real-life applications were the motivation key while evaluating the related methodology successfulness, and later they were golden guidelines when developing our own design flow within a seamless environment.

Following the architectural classification in section 2, our targeted exploration architecture fits into the second group of co-design exploration suitable architectures (processor, accompanied with array of programmable logic). The platform is based on the Intel Strong ARM microprocessor, supporting a variety of peripheries which eases communication and extends its flexibility /14/. HW programmability is achieved by introducing FPGA connected directly onto the microprocessor's bus. The platform supports the Linux operating system. Integration of SW executive and necessary control logic with the rest of programmable HW resources is supported through kernel drivers. While improving the design flow, we also made a move towards the PC based HW/SW co-design platform /15/, presented as the third possible architecture in section 2.

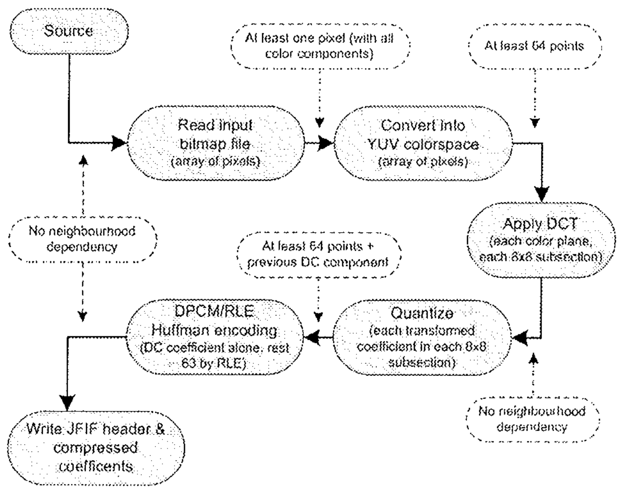
Source
At least one pixel (with all color components)
At least 64 points
Read input bitmap file (array of pixels)
Convert into YUV colorspace (array of pixels)
No neighbourhood dependency
At least 64 points + previous DC component
Apply DCT (each color plane, each 8x8 subsection)
DPCM/RLE Huffman encoding (DC coefficient alone, rest 63 by RLE)
Quantize (each transformed coefficient in each 8x8 subsection)
No neighbourhood dependency
Write JFIF header & compressed coefficents

*Figure 3: JPEG encoding steps*

In order to carefully study the whole design flow, we decided to manually realize all design steps by implementing the JPEG coding / decoding image processing application

/17/. The JPEG image processing overall complexity is well suited for a wide range of processors and their performance improvement can be substantial, provided that an optimal HW-SW co-design solution is found. Figure 3 shows JPEG compression steps, suitable as a starting point for the system level application description. This figure also exhibits the high level input application specification tightly connected to our basic idea of the high level task description. A closer examination of encoding steps already reveals not easily observable possibilities of design decisions, explicitly at the system level. Although the compression flow exhibits pure sequential data processing, the exploration space can be still revealed. Shaded nodes in Figure 3 straightforwardly resemble the JPEG compression flow. Only broken line circuited explanations commenting data dependency are here important. Dependency comments explain the minimally required amount of data provided by the previous task and needed to start the next one. If every task requirements were an entire image array, then the application task flow would be seen as strictly sequential. Here, we can discover/apply a mixed sequential, parallel, and in some stages even pipelined behavior. An optimal system level partitioning and scheduling decision would consider the required/available amount of memory between tasks, number of necessary task repetitions linked to the task execution time, and possible resource conflicts. Figure 3 expresses a close resemblance to DAG; the application is logically coarsely partitioned into subtasks, represented with nodes, and tasks data dependency is represented with arrows. After the tasks resource usage is evaluated, guidelines for finer task re-partitioning are obtained, leading into a successively refined partitioning scheme.

Following the Figure 2 design flow, a coarse grain architecture description is provided by means of a gcc compiler and gdb debugger, limitation of only one SW processor, and a certain amount of the available memory. A detailed library description is provided by a Wishbone compliant 1D DCT core, Wishbone communication structure, and device driver (SW/processor to HW/FPGA communication). Since we chose a point from the design space manually, the only constraint that makes sense is HW and SW resource usage limitation.

As every portion of the design flow was processed manually and the turnaround time was expected to be rather long, our focus was not on design space exploration, but rather on realization of the entire design flow from description to realization. It was noted that by choosing just one design point from the design space, only a suboptimal solution could be obtained. The design space is explored thoroughly as the entire flow is automated thus shortening the time required for the design decision.

Rather than generating an executable application description for several partitioning and scheduling schemes with the use of SystemC, we analyzed application execution by hand C coding. By studying the JPEG coding algorithm, it was easily established that DCT is computationally the most intensive part of image processing. We decided to implement DCT in HW (VHDL) and the rest of application sequentially in SW (C). Considering our modest initial constraints, we were successful. However, despite straightforward partitioning scheme, it took us some time to hand write the necessary code. Any modification at the system level required from us a fair amount of tedious handwriting. The main drawback of the ad-hoc approach is that a lot of handwriting has to be done. Namely, not taking the entire system integrity (e.g. communication resource conflicts) into account makes the partitioning scheme inefficient. Solution to this problem is automatization of evaluating successively chosen design points.

## 6 Conclusion

We presented a HW/SW co-design methodology design flow based on the study of current research activities in this area and our experiences with an ad-hoc approach to partitioning and system level integration. With the presented ad-hoc approach drawbacks we highlighted features that we found particularly important. We proposed a seamless environment supporting system level development and automatization of repetitive tasks, and softened the solution to the issue of heterogeneity gap. The main idea is to reuse the already existent tools. Our future research effort will be towards blurring the gap caused by languages capable of describing heterogeneous capabilities.

Our future work will address the following two main issues. First, the persistent need of increasing the system complexity will widen the interest in IP-cores reusability. Our effort will be laid in finding an efficient way of IP libraries re-usage leveraged by user code development. IP libraries can consist of any of fine grain and coarse grain operators, encapsulating wrappers enabling automatic application subtasks connectivity, and communication channels. The second area of our future work will be towards optimizing specific parts of the design flow.

## 7 References

/1/ *R. Hartenstein,* Data-stream-based Computing: Models and Architectural Resources, MIDEM Proceedings 2003, October 2003, pp. 29-37

/2/ *G. Stitt, R. Lysecky, F. Vahid,* Dynamic Hardware/ Software Partitioning: A First Approach, Design Automation Conference Proceedings 2003 (DAC'03), pp. 250-255

/3/ *M. Akil,* High-level Synthesis based upon Dependence Graph for Multi-FPGA, MIDEM Proceedings 2003, October 2003, pp. 83-96

/4/ *T. Grandpierre, C. Lavarenne and Y. Sorel,* Optimized Rapid Prototyping For Real Time Embedded Heterogeneous Multiprocessors, CODES'99 7th International Workshop on Hardware/ Software Co-Design, Rome, May 1999

/5/ *F. Vahid, T. Givargis,* Embedded System Design - A Unified Hardware/Software Introduction, John Wiley & Sons, Inc., 2002, ISBN 0-471-38678-2

/6/ *T. Wiangtong, P. Y. K. Cheung, and W. Luk,* A Unified Codesign Run-time Environment for the UltraSONIC Reconfigurable Computer, Field-Programmable Logic and Applications Proceedings, September 2003, pp. 396-405

/7/ *T. Wiangtong, Peter Y.K. Cheung, W. Luk*, Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign, International Journal on Design Automation for Embedded Systems, Kluwer Academic Publishers, Volumn 6, pp.425-449, 2002

/8/ *G. De Micheli,* Synthesis and optimization of digital circuits, McGraw-Hill, Inc., 1994, ISBN 0-07-016333-2

/9/ *J. J. Resano, M. E. Perez, D. Mozos, H. Mecha, J. Septien,* Analyzing communication overheads during hardware/software partitioning, Microelectronics Journal 34 (2003), pp. 1001 - 1007

/10/ SystemC OSCI homepage: http://www.systemc.org/

/11/ Cadence TestBuilder homepage: http://www.testbuilder.net/

/12/ Xilinx homepage: http://www.xilinx.com/

/13/ Altera homepage: http://www.altera.com/

/14/ *J. Dedič, T. Žontar, A. Janhar, A. Trost,* Design and implementation of customized embedded platform, MIDEM Proceedings, 2002, pp. 213-218

/15/ *J. Dedič, A. Trost, A. Žemva*, PC based HW/SW codesign support for embedded system target, MIDEM Proceedings, 2003, pp. 249-254

/16/ *V. D. Zivkovic, P. Lieverse,* An Overview of Methodologies and Tools in the Field of System-level Design, Lecture Notes in Computer Science, Tutorial on Embedded Processor Design Challenges, Systems, Architectures, Modeling, and Simulation (SAMOS'02), pp. 74-89, 2002

/17/ *K. Wallace*, The JPEG Still Picture Compression Standard, Communications of the ACM, April 1991, vol. 34, no. 4, pp. 30-44

*Jože Dedič, B.Sc.,*
*joze.dedic@fe.uni-lj.si,*
*Assistant Prof., Dr. Andrej Trost,*
*andrej.trost@fe-uni-lj.si*
*Associate Prof., Dr. Andrej Žemva,*
*andrej.zemva@fe-uni-lj.si*

*University of Ljubljana*
*Faculty of Electrical Engineering,*
*Tržaška 25, 1000 Ljubljana, Slovenia*
*Tel: +361 1 4768 351*

*Prispelo (Arrived): 11.02.2004* *Sprejeto (Accepted): 25.02.2004*