Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 71043, 14 pages
doi:10.1155/2007/71043

Research Article

System-Platforms-Based SystemC TLM Design of Image Processing Chains for Embedded Applications

Muhammad Omer Cheema,[1, 2] Lionel Lacassagne,[2] and Omar Hammami[1]

[1] *EECS Department, Ecole Nationale Superieure de Techniques Avancees, 32 Boulevard Victor, 75739 Paris, France*
[2] *Axis Department, University of Paris Sud, 91405 Orsay, France*

Received 18 October 2006; Accepted 3 May 2007

Recommended by Paolo Lombardi

Intelligent vehicle design is a complex task which requires multidomains modeling and abstraction. Transaction-level modeling (TLM) and component-based software development approaches accelerate the process of an embedded system design and simulation and hence improve the overall productivity. On the other hand, system-level design languages facilitate the fast hardware synthesis at behavioral level of abstraction. In this paper, we introduce an approach for hardware/software codesign of image processing applications targeted towards intelligent vehicle that uses platform-based SystemC TLM and component-based software design approaches along with HW synthesis using SystemC to accelerate system design and verification process. Our experiments show the effectiveness of our methodology.

1. INTRODUCTION

Embedded systems using image processing algorithms represent an important segment of today's electronic industry. New developments and research trends for intelligent vehicles include image analysis, video-based lane estimation and tracking for driver assistance, and intelligent cruise control applications [1–5]. While there has been a notable growth in the use and application of these systems, the design process has become a remarkably difficult problem due to the increasing design complexity and shortening time to market [6]. A lot of work is being done to propose the methodologies to accelerate automotive system design and verification process based on multimodeling paradigm. This work has resulted in a set of techniques to shorten the time consuming steps in system design process. For example, transaction-level modeling makes system simulation significantly faster than the register transfer level. Platform-based design comes one step forward and exploits the reusability of IP components for complex embedded systems. Image processing chain using component-based modeling shortens the software design time. Behavioral synthesis techniques using system-level design languages (SLDLs) accelerate the hardware realization process. Based on these techniques, many tools have been introduced for system-on-chip (SoC) designers that allow them to make informed decisions early in the design process which can be the difference in getting products to market quicker. The ability to quickly evaluate the cross-domain effects of design tradeoffs on performance, power, timing, and die size gives a huge advantage much earlier than was ever achievable with traditional design techniques.

While time to market is an important parameter for system design, an even more important aspect of system design is to optimally utilize the existing techniques to meet the computation requirements of image processing applications. Classically, these optimization techniques have been introduced at microprocessor level by customizing the processors and generating digital signal processors, pipelining the hardware to exploit instruction-level parallelism, vectorizing techniques to exploit data-level parallelism, and so forth. In system-level design era, more emphasis has been on the techniques that are more concerned with interaction between multiple processing elements instead of optimization of individual processing elements, that is, heterogeneous MPSoCs. HW/SW codesign is a key element in modern SoC design techniques. In a traditional system design process, computation intensive elements are implemented in hardware which results in the significant system speedup at the cost of increase in hardware costs.

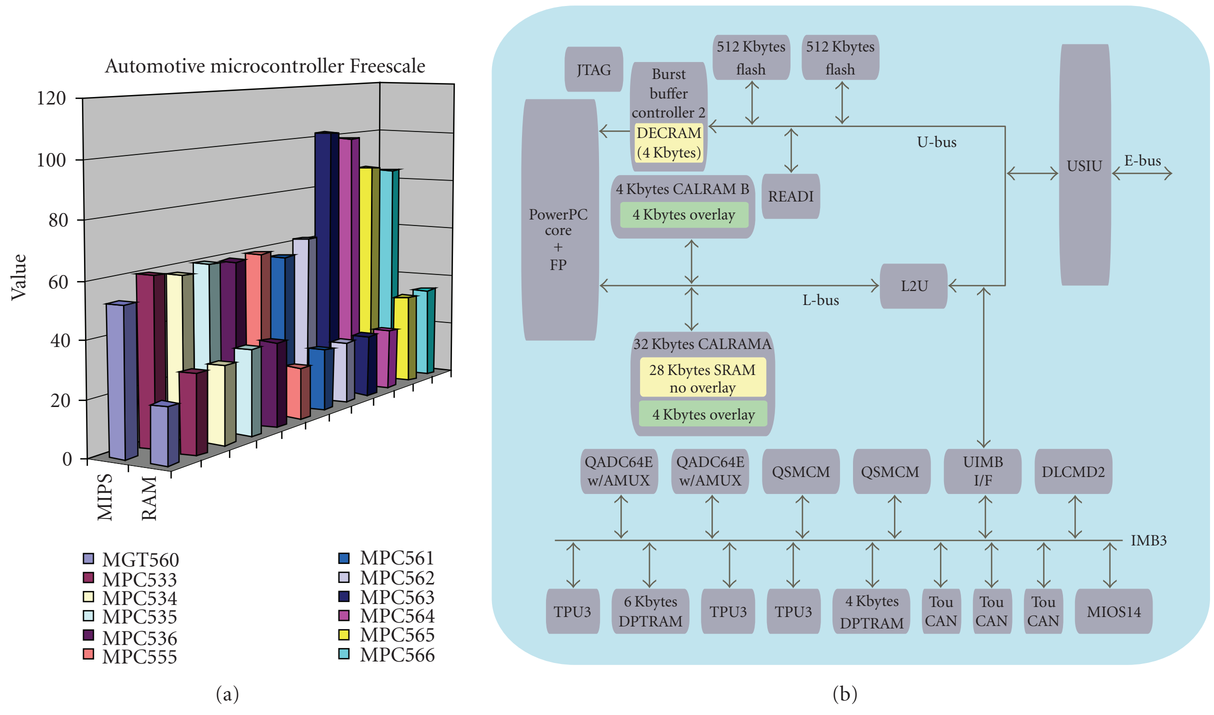


Figure 1: Freescale MPC controllers: (a) MIPS/embedded RAM, (b) MPC 565 block diagram.

In this paper, we propose an HW/SW codesign methodology that advocates the use of the following.

(i) Platform-based transaction-level modeling to accelerate system-level design and verification.
(ii) Behavioral synthesis for fast hardware modeling.
(iii) Component-based SW development to accelerate software design.

Using these techniques, we show that complex embedded systems can be modeled and validated in short times while providing satisfactory system performance.

Rest of the paper is organized as follows. Section 2 presents related work. Section 3 overviews the general vehicle design methodology and establishes a direct link with our proposal. Section 4 describes a very recent SystemC TLM platform: the IBM PowerPC evaluation kit. Section 5 explains our system design methodology and Section 6 describes the experiment environment and results. Future work and a proposed combined UML-SystemC TLM platform are described in Section 7. Finally, Section 8 concludes.

2. RELATED WORK

When designing embedded applications for intelligent vehicles a whole set of microcontrollers are available. An example of such an offer comes from Freescale [7] with their PowerPC-based microcontrollers (Figure 1).

However, although diverse in the MIPS and embedded RAM these microcontrollers do not offer enough flexibility to add specific hardware accelerators such as those required by image processing applications. The PowerPC core of these microcontrollers is not sufficient in this peripherals intensive environment to exclusively support software computation intensive applications. It is then necessary to customize these microcontrollers by adding additional resources while keeping the general platform with its peripherals. A system-design approach is needed. Our work is based on three different aspects of system design. Although some work has been done on each of these aspects at individual level, no effort has been made to propose a complete HW/SW codesign flow that gets benefit out of all these techniques to improve the system productivity. In the following sections, we will present the related work done on each of these domains. Transaction-level modeling based on system-level design languages has proven to be a fast and efficient way of system design [8–10]. It has been shown that simulation at this level is much faster [8] than register transfer level (RTL) and makes it possible for us to explore the system design space for HW/SW partitioning and parameterization. The idea of transaction-level modeling (TLM) is to provide in an early phase of the hardware development transaction-level models of the hardware. Based on this technique, a fast-enough simulation environment is the basis for the development of hardware and hardware dependent software. The presumption is to run these transaction-level models at several tens or some hundreds of thousand transactions per second which should be fast-enough for system-level modeling and verification. A lot of work has been done on behavioral synthesis. With the evolution of system-level design languages, the interest in efficient hardware synthesis based on behavioral description of a hardware module has also been visible. A few tools for behavioral SystemC synthesis [11, 12] are available in the market.

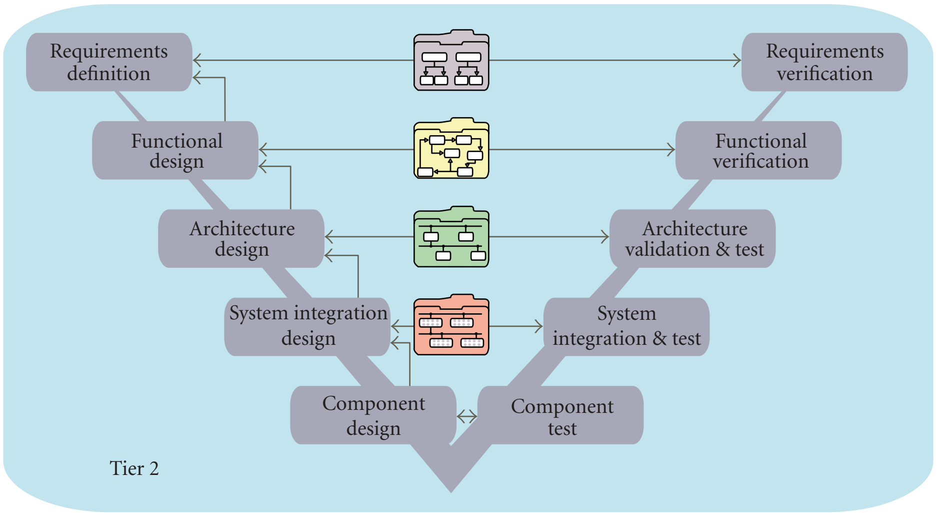


FIGURE 2: V design cycle.

For a system designer, behavioral system is very attractive for hardware modeling as it has shown to result in a lot of productivity improvements [10]. On the other hand, image processing chain development is a relatively old technique for software development that uses component-based software design to accelerate the software development process [13, 14]. On another side, UML-based design flows [15–21] have been proposed whether or not with SystemC [22–27] as an approach for fast executable specifications. However, to the best of our knowledge no tools have been proposed which combine UML- and SystemC TLM-based platforms. In this regard, additional work remains to be done in order to obtain a seamless flow.

3. GENERAL VEHICLE DESIGN METHODOLOGY

Vehicle design methodology follows the V-cycle model where from a requirements definition the process moves to functional design, architecture design, system-integration design, and component design before testing and verifying the same steps in reverse chronological order (Figure 2).

In the automotive domain, system integrator (car manufacturers) collaborate with system designer (tier 1 supplier, e.g., Valeo) while themselves collaborate with component designers (tier 2 supplier, e.g., Freescale); see (Figure 3).

This includes various domains such as electronics, software, control, and mechanics. However, design and validation requires a modeling environment to integrate all these disciplines. Unfortunately, running a complete multidomain exploration through simulation is unfeasible. Although component reuse helps somewhat reduce the challenge, it prevents from all the possible customizations existing in current system-on-chip design methodologies. Indeed, system on chip makes intensive uses of various IPs and among them parametrizable IPs which best fit the requirements of the application. This allows new concurrent design methodologies between embedded software design, architecture, intermicrocontroller communication and implementation. This flattening of the design process can be best managed through platform-based design at the TLM level.

4. PLATFORM-BASED TLM DESIGN PROCESS

Platforms have been proposed by semiconductor manufacturers in an effort to ease system-level design and allow system designers to concentrate on essential issues such as hardware-software partitioning, system parameters tuning, and design of specific hardware accelerators. This makes the reuse of platform-based designs easier than specific designs.

4.1. Platforms and IBM platform driven design methodology

The IBM CoreConnect platform [28] described in Figure 4 allows the easy connection of various components, system core, and peripheral core to the CoreConnect bus architecture.

It also includes IPs of PLB to OPB and OPB to PLB bridges and direct memory access (DMA) controller, OPB-attached external bus controller (EBCO), universal asynchronous receiver/transmitter (UART), universal interrupt controller (UIC), and double data rate (DDR) memory controller. Several other peripherals are available among them CAN controllers. The platform does not specify a specific processor core although IBM family of embedded PowerPC processors connection is straightforward. This platform which mainly specifies a model-based platform have all associated tools and libraries for quick ASIC or FPGA platform design. System core and peripheral core can be any type of user-designed components whether hardware accelerators or specific peripherals and devices.

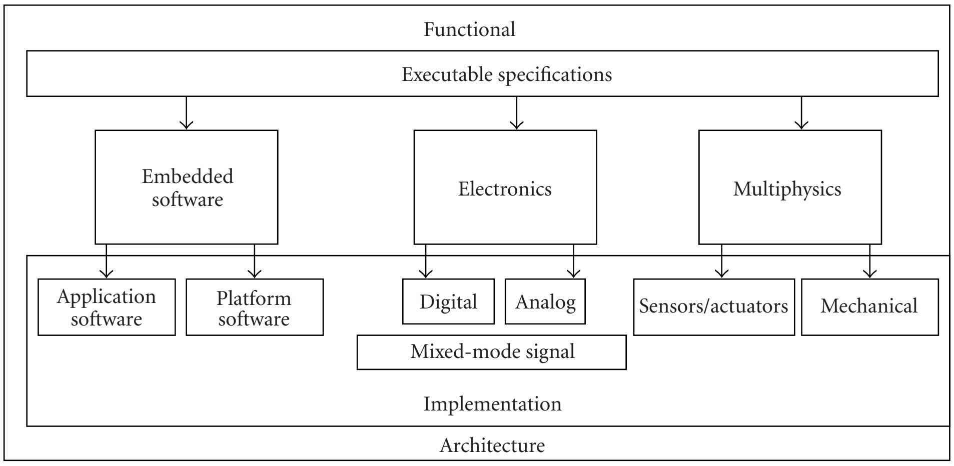


Figure 3: Decomposition.

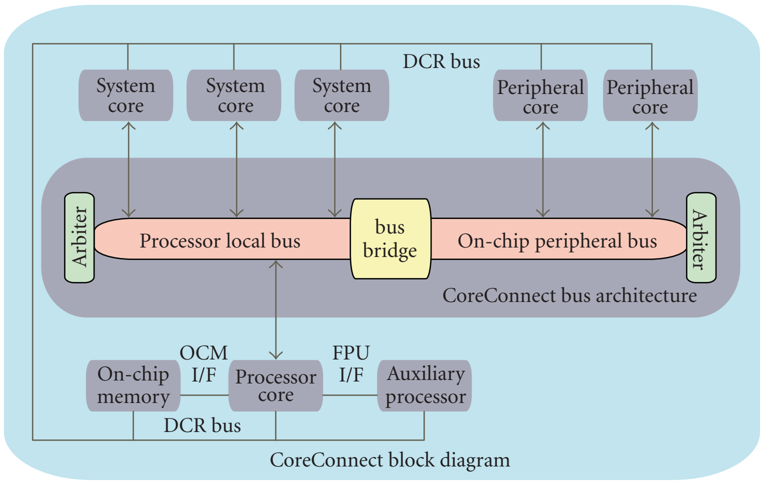


Figure 4: IBM CoreConnect platform.

4.2. *IBM SystemC TLM platform*

The SystemC IEEE standard [29] is a system-level modeling environment which allows the design of various abstraction levels of systems (Figure 5). It spawns from untimed functional to cycle accurate. In between, design space exploration with hardware-software partitioning is conducted with timed functional level of abstraction. Using the model-driven architecture (MDA) terminology [30] we can model computation independent model (CIM), platform-independent model (PIM), and platform-specific model (PSM). Besides, SystemC can model hardware units at RTL level and be synthesizable for various target technologies using tools such as Synopsys [11] and Celoxica [12], which in turn allows multiobjective SystemC space exploration of behavioral synthesis options on area, performance, and power consumption [31] since for any system, all three criteria cannot be optimally met together.

This important point allows SystemC abstraction-level platform-based evaluation taking into account area and energy aspects, and this for proper design space exploration with implementation constraints. In addition to these levels of abstraction, transaction-level modeling and abstraction level [8, 9] have been introduced to fasten simulation of communications between components by considering communications exchange at transaction level instead of bus cycle accurate levels. Benefits of TLM abstraction-level design have been clearly demonstrated [8, 9].

Using the IBM CoreConnect SystemC modeling environment PEK [32], designers are able to put together SystemC models for complete systems including PowerPC processors, CoreConnect bus structures, and peripherals. These models may be simulated using the standard OSCI SystemC [29] runtime libraries and/or vendor environments. The IBM CoreConnect SystemC modeling environment TLM platform models and environment provide designers with a

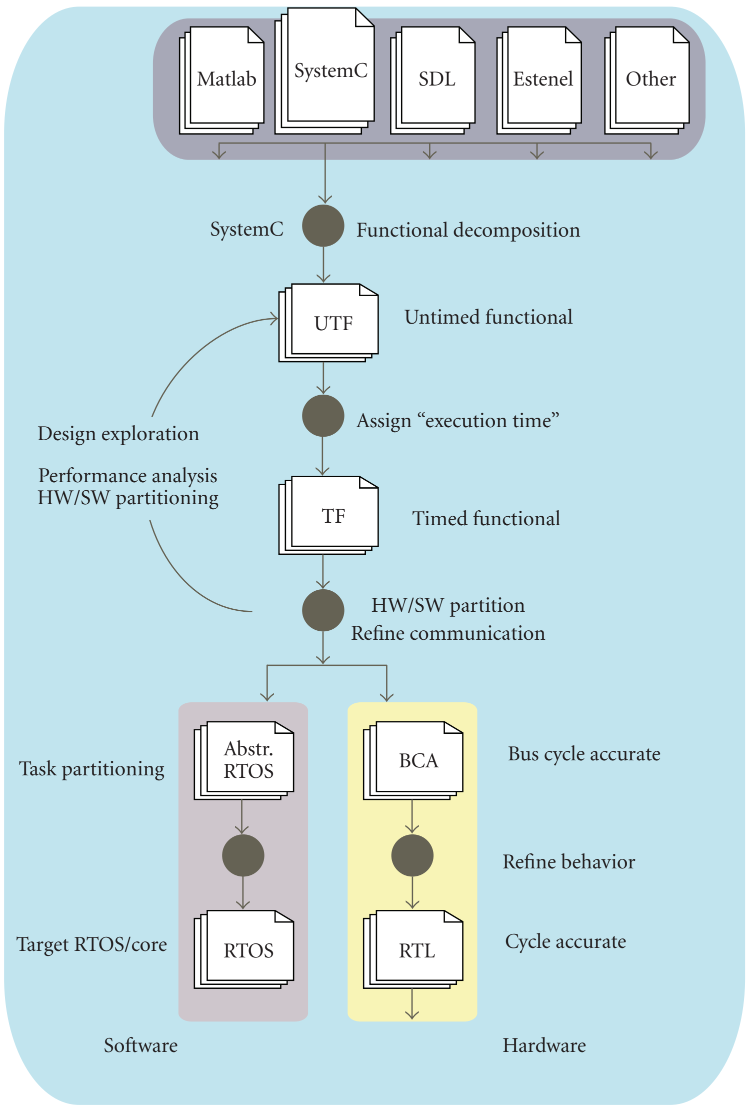


FIGURE 5: SystemC system design flow.

system simulation/verification capability with the following characteristics.

(i) Simulate real application software interacting with models for IP cores and the environment for full system functional and timing verification possibly under real-time constraints.

(ii) Verify that system supports enough bandwidth and concurrency for target applications.

(iii) Verify core interconnections and communications through buses and other channels.

(iv) Model the transactions occurring over communication channels with no restriction on communication type.

These objectives are achieved with additional practical aspects such as simulation performance must be enough to run a significant software application with an operating system booted on the system. In addition, the level of abstraction allows the following.

(i) Computation (inside a core) does not need to be modeled on a cycle-by-cycle basis, as long as the input-output delays are cycle-approximate which implies that for hardware accelerators both SystemC and C are allowed.

(ii) Intercore communication must be cycle-approximate, which implies cycle-approximate protocol modeling.

(iii) The processor model does not have to be a true architectural model; a software-based instruction set simulator (ISS) can be used, provided that the performance and timing accuracy are adequate.

In order to simulate real software, including the initialization and internal register programming, the models must be "bit-true" and register accurate, from an API point of view.

That is, the models must provide APIs to allow programming of registers as if the user were programming the real hardware device, including the proper number of bits and address offsets. Internal to the model, these "registers" may be coded in any way (e.g., variables, classes, structs, etc.) as long as their API programming makes them look like real registers to the users. Models need not be a precise architectural representation of the hardware. They may be behavioral models as long as they are cycle-approximate representations of the hardware for the transactions of interest (i.e., the actual transactions being modeled). There may be several clocks in the system (e.g., CPU, PLB, OPB). All models must be "macro synchronized" with one or more clocks. This means that for the atomic transactions being modeled, the transaction boundaries (begin and end) are synchronized with the appropriate clock. Inside an atomic transaction, there is no need to model it on a cycle-by-cycle basis. An atomic transaction is a set of actions implemented by a model, which once started, is finished, that is, it cannot be interrupted. Our system-design approach using IBM's PowerPC 405 evaluation kit (PEK) [32] allows designers to evaluate, build, and verify SoC designs using transaction-level modeling. However, PEK does not provide synthesis (area estimate) or energy consumption tools.

4.2.1. *SW development, compilation, execution, debugging*

In PEK, the PowerPC processors (PPC 405/PPC450) are modeled using an instruction-set simulator (ISS). The ISS is instantiated inside a SystemC wrapper module, which implements the interface between the ISS and the PLB bus model. The ISS runs synchronized with the PLB SystemC model (although the clock frequencies may be different). For running a software over this PowerPC processor, code should be written in ANSI C and it should be compiled using GNU cross compiler for PowerPC architecture.

The ISS works in tandem with a dedicated debugger called RiscWatch (RW) [33]. RW allows the user to debug the code running on the ISS while accessing all architectural registers and cache contents at any instance during the execution process.

4.2.2. *HW development, compilation, execution, monitoring*

Hardware modules should be modeled in SystemC using the IBM TLM APIs. Then these modules can be added to the platform by connecting them to the appropriate bus at certain addresses which were dedicated in software for these hardware modules. Both, synthesizable and nonsynthesizable SystemC can be used for modeling of hardware modules at this level but for getting area and energy estimates, it is important that SystemC code be part of standard SystemC synthesizable subset draft (currently under review by the OSCI synthesis working group) [34]. If we want to integrate already existing SystemC hardware modules, wrappers should be written that wrap the existing code for making it compatible with IBM TLM APIs. We have written generic interfaces which provide a generalized HW/SW interface hence reducing the modeling work required to generate different interfaces for every hardware module based on its control flow.

For simulation of SystemC, standard systemc functionality can be used for .vcd file generation, bus traffic monitoring and other parameters. We have also written the dedicated hardware modules which are connected with the appropriate components in the system and provide us with the exact timing and related information of various events taking place in the hardware environment of the system.

4.2.3. *Creating and managing transactions*

In a real system, tasks may execute concurrently or sequentially. A task that is executed sequentially, after another task, must wait till the first task has completed before starting. In this case, the first task is called a blocking task (transaction). A task that is executed concurrently with another need not wait for the first one to finish before starting. The first task, in this case, is called a nonblocking task (transaction).

Transactions may be blocking or nonblocking. For example, if a bus master issues a blocking transaction, then the transaction function call will have to complete before the master is allowed to initiate other transactions. Alternatively, if the bus master issues a nonblocking transaction, then the transaction function call will return immediately, allowing the master to do other work while the bus completes the requested transaction. In this case, the master is responsible for checking the status of the transaction before being able to use any result from it. Blocking or nonblocking transactions are not related to the amount of data being transferred or to the types of transfer supported by the bus protocols. Both multibyte burst transfers as well as single-byte transfers may be implemented as blocking or nonblocking transactions.

When building a platform, the designer has to specify the address ranges of memory and peripherals attached to the PLB/OPB busses. The ISS, upon encountering an instruction which does a load/store to/from a memory location on the bus, will call a function in the wrapper code which, in turn, issues the necessary transactions on the PLB bus. The address ranges of local memory, bus memory, cache sizes, cacheable regions, and so forth, can all be configured in the ISS and the SystemC models.

4.2.4. *IP parameterization*

Various parameters can be adjusted for the processor IPs and other IPs implemented in the system. For a processor IP, when the ISS is started, it loads a configuration file which contains all the configurable parameters for running the ISS. The configuration file name may be changed in the Tcl script invoking the simulation. The parameters in the file allow the setting of local memory regions, cache sizes, processor clock period, among other characteristics. For example, we can adjust the value of data and Instruction Cache sizes to be 0, 1024, 2048, 4096, 8192, 16384, 32768, and 65536 for the 405 processor. Besides setting the caches sizes, the cache regions need to be configured, that is, the user needs to specify which memory regions are cacheable or not. This is done by setting appropriate values into special purpose registers DCCR and ICCR. These are 32-bit registers, and each bit must be set to 1 if the corresponding memory region should be cacheable

The PowerPC uses two special-purpose registers (SPRs) for enabling and configuring interrupts. The first register is the machine state register (MSR) which controls processor core functions such as the enabling and disabling of interrupts and address translation. The second register is the exception vector prefix register (EVPR). The EVPR is a 32-bit register whose high-order 16 bits contain the prefix for the address of an interrupt handling routine. The 16-bit interrupt vector offsets are concatenated to the right of the high-order bits of the EVPR to form the 32-bit address of an interrupt handling routine. Using RiscWatch commands and manipulating startup files to be read from RiscWatch, we can enable/disable cachebility, interrupts, and vary the cache sizes. While on the other hand, CPU, bus, and hardware IP configuration-based parameters can be adjusted in top level file for hardware description where the hardware modules are being initialized.

Provision of these IPs and ease of modeling makes IBM TLM a suitable tool for platform generation and its performance analysis early in the system design cycle.

5. PROPOSED METHODOLOGY

It should be clear from Section 4 that IBM PEK provides almost all important aspects of system design. That is why we have based our methodology for HW/SW codesign on this tool. However, our methodology will be equally valid for all other tools having similar modeling and simulation functionality. Our HW/SW codesign approach has the following essential steps.

(a) Image processing chain development.
(b) Software profiling.
(c) Hardware modeling of image processing operators.
(d) Performance/cost comparison for HW/SW implementations.
(e) Platform generation, system design space exploration.

(a) Image processing chain development

Our system codesign approach starts from development of image processing chain (IPC). Roughly speaking, an image processing chain consists of various image processing operators placed in the form of directed graph according to the data flow patterns of the application. An image processing chain is shown in Figure 6.

This IPC describes the working of a Harris corner detector. IPC development process is very rapid as normally most of the operators are already available in the operator's library and they need only to be initialized in a top-level function to form an image processing chain and secondly it provides a very clean and modular way to optimize various parts of the application without the need of thorough testing and debugging. In our case, we have used coding guidelines as recommended by numerical recipes [35] which simplifies the IPC development process even further.

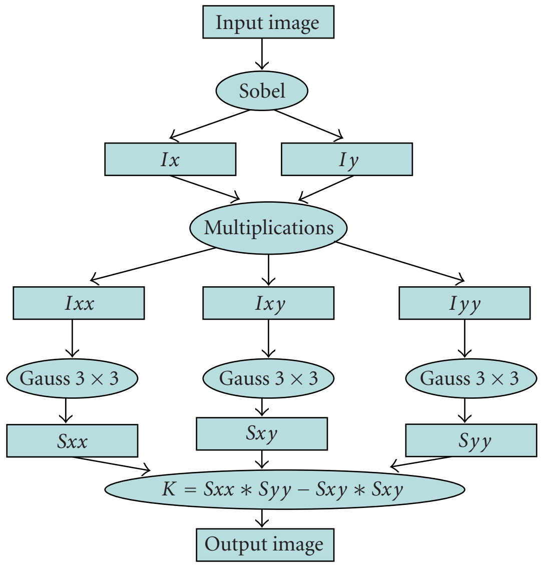


Figure 6: Harris corner detector chain.

(b) Software profiling

In this step, we execute the image processing chain over the PowerPC 405 IP provided with PowerPC evaluation kit. Using RisCWatch commands, we get the performance results of various software components in the system and detect the performance bottlenecks in the system. Software profiling is done for various data and instruction caches sizes and bus widths. This information helps the system designer take the partitioning decisions in later stages.

(c) Hardware modeling of image processing operators

In the next step of our system design approach, area and energy estimates are obtained for the operators implemented in the image processing chain. At SystemC behavioral level, the tools for estimating area and energy consumption have recently been showing their progress in the EDA industry. We use Celoxica's agility compiler [12] for area estimation in our case but our approach is valid for any behavioral-level synthesis tool in the market. As we advocate the fast chain development through libraries containing image processing operators, similar libraries can also be developed for equivalent SystemC image processing operators which will be reusable over a range of projects hence considerably shortening the hardware development times as well. At the end of this step, we have speed and area estimates for all the components of the image processing chain to be synthesized. This information is stored in a database and is used during HW/SW partitioning done in the next step.

Another important thing to be noted is that HW synthesis is also a multiobjective optimization problem. Previously,

[31] have worked over efficient HW synthesis from SystemC and shown that for a given SystemC description, various HW configurations can be generated varying in area, energy, and clock speeds. Then the most suitable configuration out of the set of pareto optimal configurations can be used in the rest of the synthesis methodology. Right now, we do not consider this HW design space exploration for optimal area/energy and speed constraints but in our future work, we plan to introduce this multiobjective optimization problem in our synthesis flow as well.

(d) Performance comparison for HW/SW implementations

At this stage of system codesign, system designer has profiling results of software as well as hardware implementation costs and the performance of the same operator in the hardware. So, in this stage performance of various individual operators is compared and further possibilities of system design are explored.

(e) Platform generation, system-design space exploration

Like traditional hardware/software codesign approaches, our target is to synthesize a system based on a general purpose processor (in our case, IBM PowerPC 405) and extended with the help of suitable hardware accelerators to significantly improve the system performance without too much increase in the hardware costs. We have chosen PowerPC 405 as a general purpose processor in our methodology because of its extensive usage in embedded systems and availability of its systemC models that provide ease of platform design based on its architecture. Our target platform is shown in Figure 7. Our target is to shift the functionality from image processing chain to the hardware accelerators such that system gets good performance improvements without too much hardware costs.

In this stage, we perform the system-level simulation. Based on the results of last step, we generate various configurations of the system putting different operators in hardware and then observing the system performance. Based on these results and application requirements, a suitable configuration is chosen and finalized as a solution to HW/SW codesign issue.

(f) Parameter tuning

In the last step of image processing chain synthesis flow, we perform the parameterization of the system. At this stage, our problem becomes equivalent to (application specific standard products) ASSP parameterization. In ASSP, hardware component of the system is fixed; hence only tuning of some soft parameters is performed for these platforms to improve the application performance and resource usage. Examples of such soft parameters include interrupt and arbitration priorities. Further parameters associated with more detailed aspects of the behavior of individual system IPs may also be available. We deal with the problem manually instead of relying on a design space exploration algorithm and our approach is to start tuning the system with the maximum resources available and keep on cutting down the resource availability until the system performance remains well within the limits and bringing down the value of a parameter does not dramatically affect system performance. However, in the future we plan to tackle this parameterization problem using automatic multiobjective optimization techniques.

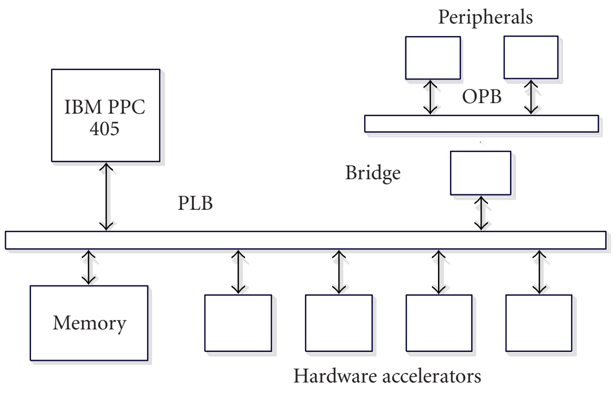


Figure 7: Target platform built using IBM TLM.

6. EVALUATION RESULTS

We have tested our approach of HW/SW codesign for Harris corner detector application described in Figure 6. Harris corner detector is frequently used for point-of-interest (PoI) detection in real-time embedded applications during data preprocessing phase.

The first step, according to our methodology, was to develop image processing chain (IPC). As mentioned in the previous section, we use numerical recipes guidelines for component-based software development and it enables us to develop/modify IPC in shorter times because of utilization of existing library elements and clarity of application flow. At this stage, we put all the components in software. Software is profiled for various image sizes and results are obtained. Next step is to implement hardware and estimate times taken for execution of an operator entirely implemented in hardware and compare it to the performance estimates of software. The results obtained from hardware synthesis and its performance as compared with software-based operations are shown in Table 1 and Figure 6.

Results in Table 1 show the synthesis results of behavioral SystemC modules for different operators computing different sizes of data. We can see that with the change in data size, memory requirements of the operator also change, while the part of the logic which is related to computation remains the same. Similarly, critical path of the system remains the same as it mainly depends on computational logic structure. Based on the synthesized frequencies and number of cycles required to perform each operation, last column shows the computation time for each hardware operator for a given size of data. It is again worth mentioning that synthesis of these operators depends largely on the intended design. For example, adding multiport memories can result in acceleration in read

TABLE 1: Synthesis results for Harris corner detector chain.

Module name	Area (computational logic and memory)			Critical path (ns)	Synth. freq. (MHz)	Total comp. time (μs)
	Size	Comp. logic slices	memory (bits)			
Sobel	8×8	218	18432	14.41	69.39	1.845
	16×16	220	18432	14.41	69.39	7.376
	32×32	222	36864	14.41	69.39	29.514
	64×64	224	131072	14.41	69.39	118.06
P2P Mul	8×8	151	36864	11.04	90.33	1.417
	16×16	151	36864	11.04	90.33	5.668
	32×32	152	73728	11.04	90.33	22.67
	64×64	152	262144	11.04	90.33	90.69
Gauss	8×8	184	18432	16.37	61.1	2.095
	16×16	186	18432	16.37	61.1	8.38
	32×32	188	36864	16.37	61.1	33.52
	64×64	190	131072	16.32	61.1	134.1
K = coarsity computation	8×8	351	36864	19.32	51.76	2.473
	16×16	352	73728	19.32	51.76	9.892
	32×32	353	147456	19.32	51.76	39.567
	64×64	354	294912	19.32	51.76	158.269

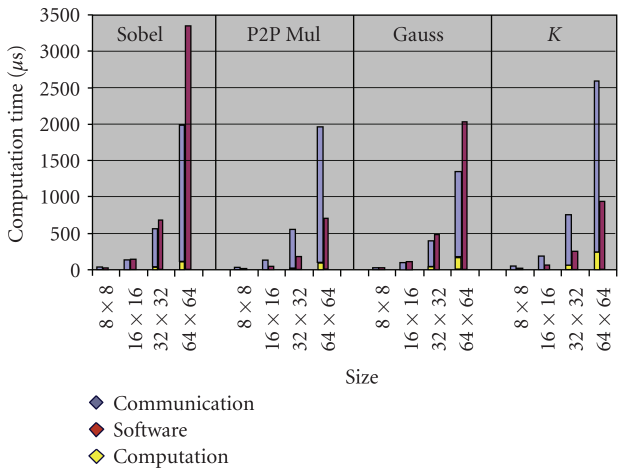


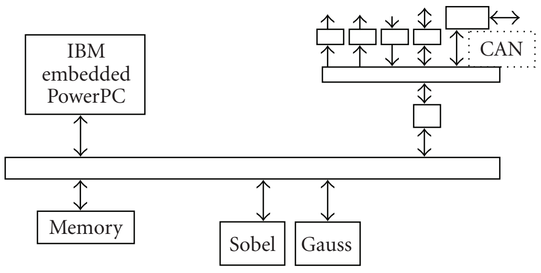
FIGURE 8: HW performance versus SW performance of operators.

operations from memory while unrolling the loops in SystemC code can result in performance improvement at a cost of an increase in area.

Figure 8 shows the comparison of execution times of an operator in its hardware and software implementations. There are two things to be noticed here. Firstly, operator computation time for hardware has been shown with two different parameters: computation and communication. Looking at Table 1, one might feel that all hardware implementations will be much faster than their software version but one needs to realize here that implementing a function in hardware requires the data to be communicated to the hardware module which requires changes in software design where computation functions are replaced by data transfer functions. Although image processing applications seem to be computation intensive, it should be noted that most of the time is taken up by communication while computation is only a fraction of total time taken by the hardware. An ideal function to be implemented in hardware will be the one which has lesser data to be transferred from/to the hardware to/from the general purpose processor. Secondly, in the example, we can see that Gaussian and Sobel operators seem to be better candidates to be put in hardware while coarsity computation in hardware lags in performance than its software version because of lesser computation and more communication requirements of the function.

After the performance comparison of operators in hardware and software, next step was to generate the platform and perform the system-level simulation for various configurations. For our system-level simulation, our general purpose processor (PowerPC 405) was running at 333 MHz while it had 16 Kbytes of data and instruction caches.

At first simulation run, we realized that due to data accesses, original software was spending a lot of time in memory access operations. We optimized the software which resulted in an optimized version of the software. After that, we started exploring HW/SW codesign options by generating various versions and getting the simulation results. Table 2 shows a few of the configurations generated and the CPU cycles taken by the system during the simulation. A quick look at the results shows that taking into consideration of hardware implementation cost, configuration 7 provides a good speedup where we have implemented Gaussian and Gradient functions in the hardware. Table 1 shows that adding these operators to hardware will result in a slight increase in computation logic while a bit more increase in memory and at that cost a speedup of more than 2.5 can be obtained.



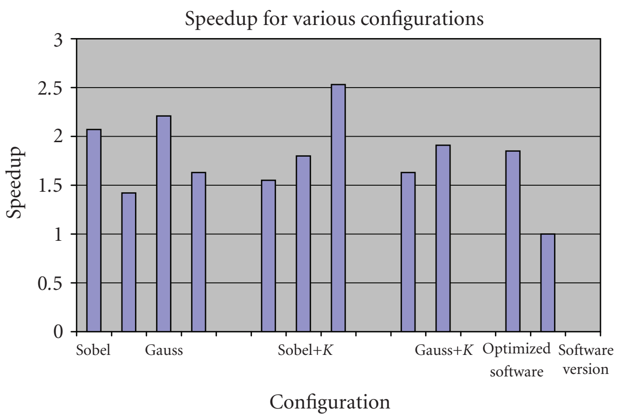


Figure 9: (a) Platform configuration 7. (b) Full HW/SW design space exploration results.

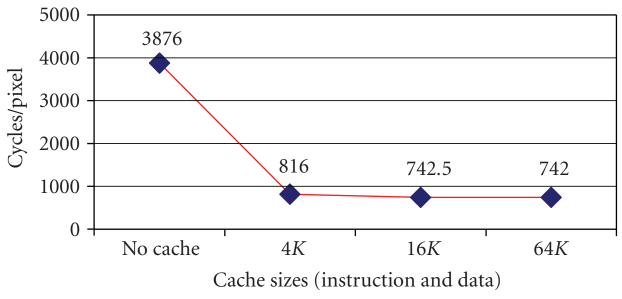


Figure 10: Various cache sizes and system performance.

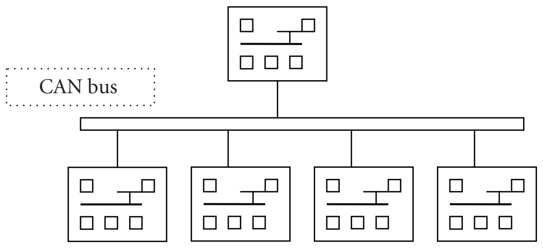


Figure 11: Platforms networked through CAN bus.

Figure 9 graphically represents Table 2. We can see that the configuration involving Sobel and Gaussian operators gives significant speedups while configurations involving point-to-point multiplication and coarsity computation (K) result in worse performance. Based on these results, a system designer might choose configuration 7 for an optimal solution. Or if he has strong area constraints, configurations 1 and 3 can be possible solutions for codesigned system.

When configuration 7 was chosen to be the suitable configuration for our system, next step was the parameterization of the system. Although parameterization involves bus width adjustment, arbitration scheme management and interrupt routine selection, for the sake of simplicity we show the results for optimal sizes of caches. Figure 10 shows the results for various cache sizes and corresponding performance improvement. We can see that cache results in significant performance improvements until $16K$ of data and instruction cache sizes. But after that, the performance improvements with respect to cache size changes reach a saturation point and there is almost no difference of performance for $16K$ and $64K$ caches in the system. Hence we choose $16K$ data and instruction caches sizes for our final system.

This approach allowed us to alleviate the problem of selecting inadequate microcontrollers for intelligent vehicle design such as those described Section 2. This process can be repeated with other applications in order to build a system based on networked platforms; see Figure 11.

Lastly, we will mention the limitations of the methodology. It should be noticed that we have chosen small image sizes for our system design. Although TLM-level simulation is much faster than RTL-level simulations, it still takes a lot of time for simulation of complex systems. Increasing the image sizes beyond 256×256 for the given example makes it increasingly difficult for exploring the design space thoroughly as it required multiple iterations of simulation for each configuration and one iteration itself takes hours or even days to complete. For larger image sizes where simulation time will dominates the system design time, RTL-level system prototyping and real-time execution over hardware prototyping boards seem to be a better idea where although system prototyping will take longer times but significant time savings can be made by preferring real-time execution over simulations. The approach of [36] can be used in this context.

7. FUTURE WORK: COMBINING UML-BASED SYSTEM-DESIGN FLOW WITH SYSTEMC TLM PLATFORM FOR INTELLIGENT VEHICLES DESIGN

The work presented so far described the potentials of SystemC TLM platform-based design for the system design of embedded applications through the customization of

Table 2: Various configurations and speedups for point-of-interest detection.

Config. no.	Hardware implement	Time (cycle)	Cycle/pixel	Speedup over software version
1	Sobel	3726350	909.75	2.07
2	P2P Mul	5419590	1323.14	1.42
3	Gauss	3490064	852.06	2.21
4	K = coarsity comp.	4725762	1153.75	1.63
5	Sobel + P2P Mul	4970836	1213.58	1.55
6	Sobel + K	4277108	1044.22	1.80
7	Sobel + Gauss	3041510	742.56	2.53
8	Gauss + P2P Mul	4734654	1155.92	1.63
9	Gauss + K	4040826	986.52	1.91
10	Optimized software	4175000	1019.29	1.85
11	Original software version	7717000	1884.03	1

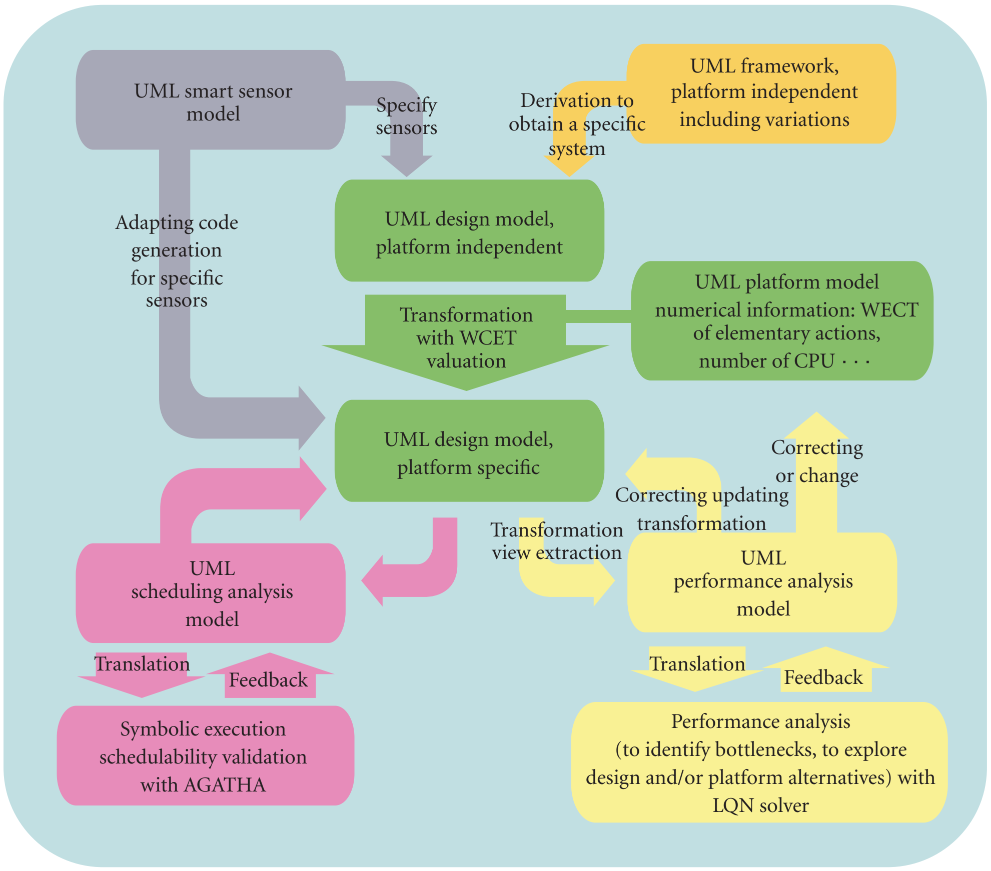

UML smart sensor model
Specify sensors
Derivation to obtain a specific system
UML framework, platform independent including variations
Adapting code generation for specific sensors
UML design model, platform independent
Transformation with WCET valuation
UML platform model numerical information: WECT of elementary actions, number of CPU · · ·
UML design model, platform specific
Correcting or change
Correcting updating transformation
Transformation view extraction
UML scheduling analysis model
UML performance analysis model
Translation
Feedback
Symbolic execution schedulability validation with AGATHA
Translation
Feedback
Performance analysis (to identify bottlenecks, to explore design and/or platform alternatives) with LQN solver

Figure 12: Accord/UML design methodology.

microcontrollers. Clearly important benefits come from this approach with the possibility to get access to implementation details (area, energy consumption) without lowering the design abstraction details down to implementation. This key point clearly contributes to the reduction of the design cycle and the ease of the design space exploration. On the other hand, several research projects have advocated the use of UML-based system design for real-time embedded systems [16–19]. The Accord/UML is a model-based methodology dedicated for the development of embedded real-time applications [16] (Figure 12). The main objectives of the methodology is to specify and prototype embedded real-time systems through three consistent and complementary models describing structure, interaction, and behavior. Examples of applications include smart transducer integration in real-time embedded systems [19].

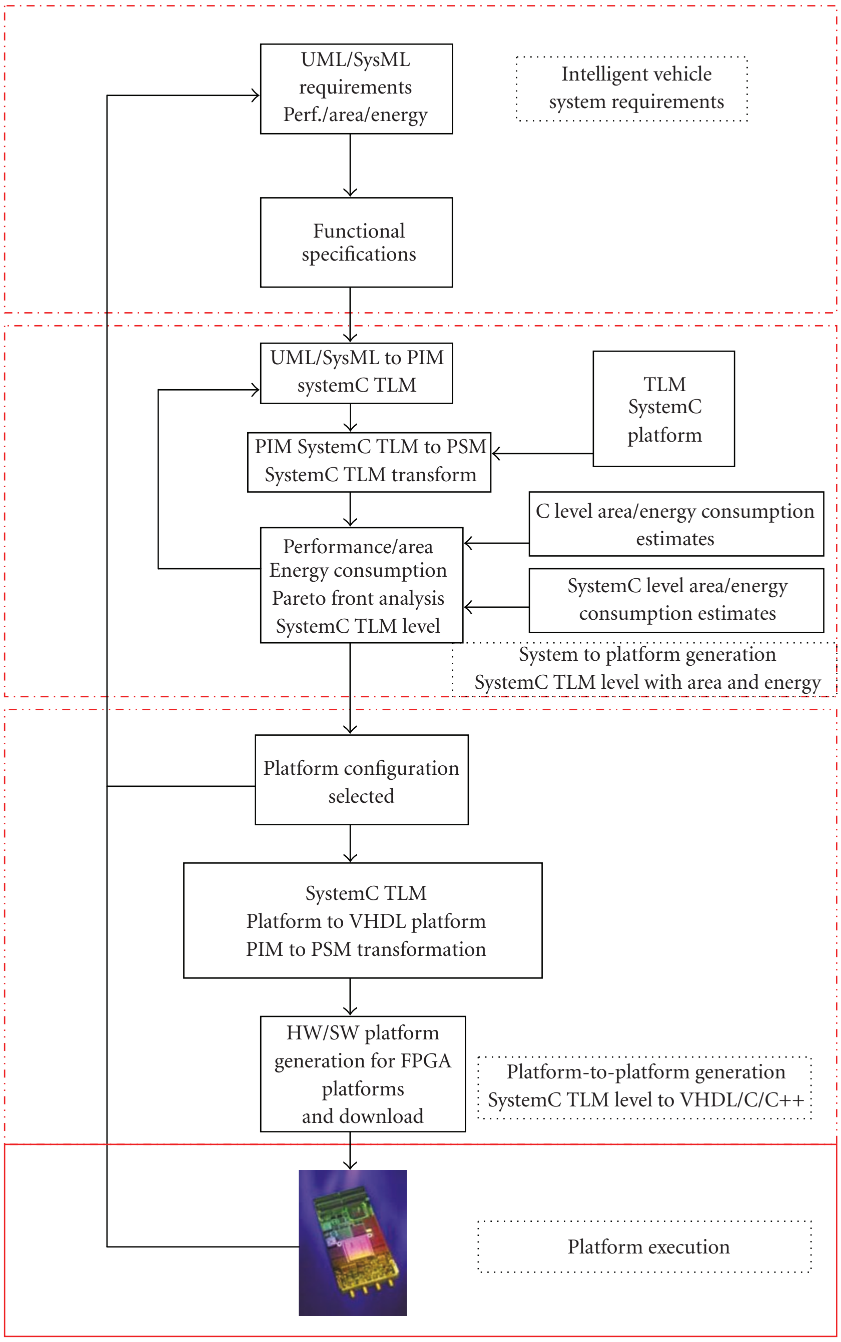

UML/SysML requirements Perf./area/energy
Intelligent vehicle system requirements
Functional specifications
UML/SysML to PIM systemC TLM
TLM SystemC platform
PIM SystemC TLM to PSM SystemC TLM transform
C level area/energy consumption estimates
Performance/area Energy consumption Pareto front analysis SystemC TLM level
SystemC level area/energy consumption estimates
System to platform generation SystemC TLM level with area and energy
Platform configuration selected
SystemC TLM Platform to VHDL platform PIM to PSM transformation
HW/SW platform generation for FPGA platforms and download
Platform-to-platform generation SystemC TLM level to VHDL/C/C++
Platform execution

Figure 13: UML/SysML/TLM SystemC platform-based design methodology for intelligent vehicles.

One key step of the Accord/UML methodology is the model transformation from a UML design model platform independent to a UML design model platform specific. This is mainly accomplished through a transformation with a worst-case execution time (WCET) valuation. This PSM could be improved by iterating through a UML performance analysis model which would again influence the transformation. This performance analysis model could be conducted using SystemC TLM platform model and include additional analysis with area and energy consumption as we did in the previous section. The objectives of the ProMARTE working group is to define a UML profile for modeling and analysis of real-time and embedded systems (MARTE) that answers to the RFP for MARTE [17]. These examples of UML-based design methodologies of embedded real-time systems suggest that UML and platform SystemC TLM design methodologies may be combined for intelligent vehicles design. In this regard, the autosar organization have released its UML profile v1.0.1 as a metamodel to describe the system, software, and hardware of an automobile [37]. This profile is expected to be used as well for intelligent vehicles design. However, translation from UML/SysML to SystemC have only recently been tackled. Work has been conducted on the description of executable platforms at the UML-level as well as the translation of UML-based application descriptions to SystemC [27]. However, this work is far from getting down to a SystemC level of the platform we used in this study. In [25] they present a UML2.0 profile of SystemC language exploiting MDA capabilities. No significant example of the methodologies is shown. In [23] a bi-directional UML-SystemC translation tool called UMLSC is described. According to the authors more work remains to be done to extend UML to make it better suited for hardware specification and improve the translation tool. In [26] translation from UML to SystemC for stream processing applications is presented. This work allows the translation of a stream processor, however, not a full-fledged processor. It is an implementation of the abstract model in UML 2.0. A very recent significant example of translation is provided in [38] using network on chip. However, all the works mentioned so far did not use (1) SystemC TLM platform-based design and (2) area and energy consumption of platform configurations.

We propose a UML/SysML to SystemC design flow methodology exclusively targeting platforms, that is, we are not interested to directly translate UML to hardware level nor we are interested to translate UML to SystemC. In a SystemC TLM, platform modules have SystemC interface but can be written with C. So UML structural parts are met with structural part of SystemC TLM platform while internal behavior of modules provided in C. This requires for area/energy consumption tradeoffs C-based synthesis and energy estimate tools such as [39]. Our proposed flow transforms UML to SystemC TLM platforms with design space exploration at SystemC TLM level for timing, area, and energy (Figure 13).

In a combined UML-SystemC design methodology, UML is used to capture the static system architecture and the high-level dynamic behavior while SystemC is used for design implementation.

The transformation of the SystemC TLM to VHDL platform is straightforward and will be described in a future publication [40]. The use of FPGA platforms allows faster prototyping especially if one considers actual intelligent vehicle driving conditions [41, 42]. This overall design flow will be the focus of future work [43].

8. CONCLUSIONS

In this paper, we have proposed a platform-based SystemC TLM system-level design methodology for embedded applications. This methodology emphasizes on components-based software design and high-level (TLM) modeling and simulation. Our proposed design flow facilitates the process of system design by higher leveling hardware modeling and behavioral synthesis of hardware modules. We have showed that using the methodology, complex image processing applications can be synthesized within very short time hence increasing the productivity and reducing overall time to market for an electronic system. The introduction of Autosar UML profile suggests the use of a combination of UML based and SystemC TLM platform-based joint methodologies. Microcontrollers customized with our approach could benefit from higher-level specification. Future work will extend to raising the design methodology abstraction level to combined UML/SysML/TLM SystemC platform design flow.

REFERENCES

[1] T. Bucher, C. Curio, J. Edelbrunner, et al., “Image processing and behavior planning for intelligent vehicles,” *IEEE Transactions on Industrial Electronics*, vol. 50, no. 1, pp. 62–75, 2003.

[2] L. Li, J. Song, F.-Y. Wang, W. Niehsen, and N.-N. Zheng, “IVS 05: new developments and research trends for intelligent vehicles,” *IEEE Intelligent Systems*, vol. 20, no. 4, pp. 10–14, 2005.

[3] J. C. McCall and M. M. Trivedi, “Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 20–37, 2006.

[4] A. P. Girard, S. Spry, and J. K. Hedrick, “Intelligent cruise-control applications: real-time, embedded hybrid control software,” *IEEE Robotics & Automation Magazine*, vol. 12, no. 1, pp. 22–28, 2005.

[5] W. van der Mark and D. M. Gavrila, “Real-time dense stereo for intelligent vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 38–50, 2006.

[6] K. D. Müller-Glaser, G. Frick, E. Sax, and M. Kühl, “Multiparadigm modeling in embedded systems design,” *IEEE Transactions on Control Systems Technology*, vol. 12, no. 2, pp. 279–292, 2004.

[7] Freescale Semiconductors, http://www.freescale.com/.

[8] F. Ghenassia, Ed., *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*, Springer, New York, NY, USA, 1st edition, 2006.

[9] L. Cai and D. Gajski, “Transaction level modeling: an overview,” in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS ’03)*, pp. 19–24, Newport Beach, Calif, USA, October 2003.

[10] N. Calazans, E. Moreno, F. Hessel, V. Rosa, F. Moraes, and E. Carara, “From VHDL register transfer level to SystemC transaction level modeling: a comparative case study,” in *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI ’03)*, pp. 355–360, Sao Paulo, Brazil, September 2003.

[11] Synopsys, “Behavioral Compiler User Guide,” Version 2003.10, 2003.

[12] Agility, http://www.celoxica.com/products/agility/default.asp.

[13] O. Capdevielle and P. Dalle, “Image processing chain construction by interactive goal specification,” in *Proceedings of the 1st IEEE International Conference Image Processing (ICIP ’94)*, vol. 3, pp. 816–820, Austin, Tex, USA, November 1994.

[14] Y. Abchiche, P. Dalle, and Y. Magnien, “Adaptative Concept Building by Image Processing Entity Structuration,” Institut de Recherche en Informatique de Toulouse IRIT, Université Paul Sabatier.

[15] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, “Model-driven development using UML 2.0: promises and pitfalls,” *Computer*, vol. 39, no. 2, pp. 59–66, 2006.

[16] Accord/UML, http://www-list.cea.fr/labos/fr/LLSP/accord_uml/AccordUML_presentation.htm.

[17] ProMARTE, http://www.promarte.org/.

[18] Protes project, http://www.carroll-research.org/.

[19] C. Jouvray, S. Gérard, F. Terrier, S. Bouaziz, and R. Reynaud, "UML methodology for smart transducer integration in real-time embedded systems," in *Proceedings of IEEE Intelligent Vehicles Symposium*, pp. 688–693, Las Vegas, Nev, USA, June 2005.

[20] S. Gérard, C. Mraidha, F. Terrier, and B. Baudry, "A UML-based concept for high concurrency: the real-time object," in *Proceedings of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '04)*, pp. 64–67, Vienna, Austria, May 2004.

[21] H. Saíedian and S. Raguraman, "Using UML-based rate monotonic analysis to predict schedulability," *Computer*, vol. 37, no. 10, pp. 56–63, 2004.

[22] J.-L. Dekeyser, P. Boulet, P. Marquet, and S. Meftali, "Model driven engineering for SoC co-design," in *Proceedings of the 3rd International IEEE Northeast Workshop on Circuits and Systems Conference (NEWCAS '05)*, pp. 21–25, Quebec City, Canada, June 2005.

[23] C. Xi, L. J. Hua, Z. ZuCheng, and S. YaoHui, "Modeling SystemC design in UML and automatic code generation," in *Proceedings of the 11th Asia and South Pacific Design Automation Conference (ASP-DAC '05)*, vol. 2, pp. 932–935, Yokohama, Japan, January 2005.

[24] J Kreku, M. Eteläperä, and J.-P. Soininen, "Exploitation oF UML 2.0—based platform service model and systemC workload simulation in MPEG-4 partitioning," in *Proceedings of the International Symposium on System-on-Chip (SOC '05)*, pp. 167–170, Tampere, Finland, November 2005.

[25] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio, "A SoC design methodology involving a UML 2.0 profile for SystemC," in *Proceedings of the Design, Automation & Test in Europe Conference (DATE '05)*, vol. 2, pp. 704–709, Munich, Germany, March 2005.

[26] Y. Zhu, Z. Sun, W.-F. Wong, and A. Maxiaguine, "Using UML 2.0 for system level design of real time SoC platforms for stream processing," in *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 154–159, Hong Kong, August 2005.

[27] K. D. Nguyen, Z. Sun, P. S. Thiagarajan, and W.-F. Wong, "Model-driven SoC design via executable UML to SystemC," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS '04)*, pp. 459–468, Lisbon, Portugal, December 2004.

[28] IBM CoreConnect, http://www.ibm.com/.

[29] *IEEE 1666 Standard SystemC Language Reference Manual*, http://standards.ieee.org/getieee/1666/download/1666-2005.pdf.

[30] MDA Guide Version 1.0.1 June 2003, OMG.

[31] S. Chtourou and O. Hammami, "SystemC space exploration of behavioral synthesis options on area, performance and power consumption," in *Proceedings of the 17th International Conference on Microelectronics (ICM '05)*, pp. 67–71, Islamabad, Pakistan, December 2005.

[32] IBM PEK v1.0, http://www-128.ibm.com/developerworks/power/library/pa-pek/.

[33] "RiscWatch Debuggers User Guide," 15th edition, IBM Number: 13H6964 000011, May 2003.

[34] OSCI SystemC Transaction-Level Modeling Working Group (TLMWG), http://www.systemc.org/web/sitedocs/technical_working_groups.html.

[35] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK, 1989.

[36] R. Ben Mouhoub and O. Hammami, "MOCDEX: multiprocessor on chip multiobjective design space exploration with direct execution," *EURASIP Journal of Embedded Systems*, vol. 2006, Article ID 54074, 14 pages, 2006.

[37] UML Profile for Autosar v1.0.1, http://www.autosar.org/.

[38] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio, "A model-driven design environment for embedded systems," in *Proceedings of the 43rd ACM/IEEE Design Automation Conference (DAC '06)*, pp. 915–918, San Francisco, Calif, USA, July 2006.

[39] Orinoco Dale, http://www.chipvision.com/company/index.php.

[40] O. Hammami and Z. Wang, "Automatic PIM to PSM Translation," submitted for publication.

[41] S. Saponara, E. Petri, M. Tonarelli, I. del Corona, and L. Fanucci, "FPGA-based networking systems for high data-rate and reliable in-vehicle communications," in *Proceedings of the Design, Automation & Test in Europe Conference (DATE '07)*, pp. 1–6, Nice, France, April 2007.

[42] C. Claus, J. Zeppenfeld, F. Müller, and W. Stechele, "Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system," in *Proceedings of the Design, Automation & Test in Europe Conference (DATE '07)*, pp. 1–6, Nice, France, April 2007.

[43] O. Hammami, "Automatic Design Space Exploration of Automotive Electronics: The Case of AUTOSAR," submitted for publication.