A FRAMEWORK FOR HIGH-LEVEL SYSTEM DESIGN EXPLORATION

Jože Dedič, Matjaž Finc, Andrej Trost

University of Ljubljana, Faculty of Electrical Engineering, Laboratory for Integrated Circuits Design, Ljubljana, Slovenia

Key words: abstraction, design-space exploration, modeling, high-level design, TLM.

Abstract: The complexity of modern embedded systems requires a revised and systematic approach to efficient and concurrent management of hardware (HW) and software (SW) parts in a codesign process. In order to optimally meet the ever increasing design requirements and at the same time leverage design productivity, higher-level aspects need to be addressed before worrying about the HW/SW boundary. This way high-level design decisions evaluation is enabled and premature ad-hoc decisions are avoided. This paper deals with high-level aspects of system-level modeling and provides modeling extension, from which contemporary related methodologies could greatly benefit. High-level aspects, their influence on the entire design flow and systematic integration into the codesign environment are presented. A design flow and realization of supporting library are detailed, both offering support for high-level exploration. Applicability of the proposed high-level codesign concepts is illustrated with a case study.

Programsko ogrodje za visokonivojsko raziskovanje na sistemskem nivoju

Kjučne besede: abstrakcija, modeliranje, raziskovanje načrtovalskega prostora, TLM, visokonivojsko načrtovanje.

Izvleček: Visoka kompleksnost modernih vgrajenih sistemov zahteva posodobljen in sistematičen način pristopa k sočasnemu načrtovanju strojne in programske opreme (SNSPO). Nenehno povečevanje načrtovalskih zahtev je ob hkratni zahtevi po povečevanju načrtovalske produktivnosti mogoče samo v primeru, da se pred obravnavo meje med strojno in programsko opremo osredotočimo na visokonivojske lastnosti sistema. To nudi osnovo za ovrednotenje visokonivojskih sistemskih odločitev, na podlagi katerih se je mogoče izogniti prezgodnjim in neutemeljenim načrtovalskim odločitvam. V članku bomo predstavili visokonivojske pristope modeliranja na sistemskem nivoju in predlagali razširjen pristop uporabe abstrakcije. Uporaba predlaganih mehanizmov je sodobnim metodologijam, ki se ukvarjajo s sorodnim raziskovanjem, lahko v veliko pomoč. Predstavljeni so visokonivojski pristopi, njihov vpliv na celoten načrtovalski potek in sistematična integracija v okolje s podporo za SNSPO. Podrobno sta prikazana načrtovalski potek razširjenega koncepta abstrakcije in izvedba podpornih knjižnic, ki omogočata visokonivojsko raziskovanje. Uporaba predlaganega visokonivojskega pristopa je prikazana na praktičnem primeru.

1 Introduction

The design complexity of the state-of-the-art systems steeply increases due to the rapidly increasing scale of integration and tightening performance demands. The major driving force behind research activities that are addressing system design is to decrease the constantly growing gap between designer productivity and increase in complexity from the underlying technology, described for example with the Moore's Law. Addressing this gap, the majority of research studies concentrate their work on various abstraction levels of HW/SW codesign /1/-/5/ and they seamlessly deal with different aspects tightly integrated. Although the designer using this approach must be fully aware of different abstraction levels (from purely abstract to implementable) and different aspects (architecture and functionality description, design space exploration, etc.), it is the role of the methodology to define the entire design flow, define all the intermediate steps of it and provide effective means for automating this extremely complex process. As a result, research efforts in HW/SW codesign are focused on seamless integration of efficient methods that the system designer would benefit from.

In this paper we will especially focus on raising the level of abstraction, which is one of the generally recognized approaches /6/-/9/. The concept of abstraction is used to effectively handle complexity at different levels of system realization, starting at system specification and ending at its implementation. Providing adequate evaluation support at system-level specification, premature system-level decisions can be avoided and the design effort can be directed towards detailed exploration of potentially feasible solutions. To enable this, support for abstraction must be leveraged and design stages must be approached systematically, while consistently following system-level optimization efforts.

Unambiguously defined abstraction levels and clearly defined transitions among them directly contribute to automation of lowering the level of abstraction /8/, /11/. Lowering the level of abstraction is based on making design decisions about the system and if the decisions to be taken are within manageable complexity and of a rigidly defined abstraction level, they could be automated. E.g. lowering the level of abstraction from C to the assembler and later to an executable binary code is supported automatically. Automation is enabled by the use of a compiler and design decisions are made by the use of switches. Yet another important reason for heightening the level of abstraction is narrowing the gap between the initial informal description of the system and formal system description

which could possibly be executed and evaluated. Currently there exists a huge gap in this area and system-level approaches that for example use UML /12/ try to cover it.

Based on the above, we propose a system-level design methodology that systematically extends the currently used levels of abstraction. For the methodology a framework implementation is explained. Section 2 classifies the types of abstraction and based on that introduces the contemporary related work. In Section 3 we will focus on transaction level modeling, which is a widely used modeling approach supporting refinements throughout different levels of abstraction. An explanation of the framework supporting extended concept of abstraction will follow in Section 4. Applicability of the extended abstraction level modeling will be demonstrated in Section 5 by means of a case study of a JPEG encoder. With conclusions and plans for future work we will end this paper.

2 Related work

Numerous research studies dealing with different aspects of the design methodology have been published /1/-/4/. While the heterogeneity issue of HW/SW codesign is still being under the examination, the research interest is slowly drifting away from the concern about HW/SW boundaries towards higher levels of abstraction. This contributes to leveraging the design productivity and enables more efficient design space exploration.

The today's widely used system-design approach dealing with an abstraction is transaction level modeling (TLM) presented by Gajski et al. in /7/. TLM abstraction is based on an approach where the design description is started by describing only the most important system-level features (specification model). Features unimportant or yet unknown for the current model of the system are simply left out (undescribed). Throughout successive refinement steps (from specification to implementation model) as the designer's knowledge of the system progresses, being the result of the model evaluation feedback, additional features are added to further detail the description of the system. This also corresponds to lowering the level of abstraction. The process of successive model refinement is finished once the model of the system is described at the abstraction level low enough for system implementation by means of automation tools. At every level of abstraction an evaluatable model (EM) is obtained. The model at the lowest level of abstraction encapsulates all the information the designer captures throughout model refinement and consists of all the necessary implementation details. Being an implementation model (IM), it provides grounds for system implementation.

Besides TLM, other types of abstraction can be identified. Jerraya et al. ground their work on the approach introducing layers of intermediate adapters /4/, /10/, /14/. The SPACE methodology applies the concept of layers of services to implement the support for the OS-like-features within SystemC /3/. This type of abstraction is based on the concept of layers of services. Interfaces provide mechanisms for connecting neighboring layers and offer efficient means of access to functionality while hiding realizations implemented within particular layers. For example, SW engineers apply programming techniques using this type of abstraction to successfully cope with complexity. Unlike the aforementioned type of abstraction, here to obtain EM and IM, all layers of abstraction must be fully implemented, as they depend on each other.

The third type of abstraction is based on the concept of platforms, presenting layers of abstraction. At each layer the designer's task is to best map the requirements with abstraction of potential implementations. Choosing a potential implementation at the current level of abstraction introduces specification of requirements for the succeeding lower level of abstraction. Similarly to the first mentioned type of abstraction, here the description of the system also starts with system-level features, but differentiates in that here individual layers of abstraction are not refined, but rather supplement each other with different levels of implementation details. Compared with abstraction based on the concept of layers of services, this type of abstraction provides EMs at all abstraction levels and does not have so rigidly defined layer's stacking. An example of the design methodology applying this concept of abstraction is Metropolis /1/, /16/ based on recursive paradigm of platform-based design /9/, /15/.

Observation of arrangements of abstraction-based approaches led us to the conclusion that these approaches belong to a set of vertical concepts for addressing complexity. They should however be thought of distinctively from a set of horizontal concepts, i.e. component-based approach /10/ and functionality-architecture separation. Horizontal concepts can coexist on all layers of abstraction. For example, the component-based approach is included as its integral part within TLM. Inspection of methodologies reveals that all the above mentioned concepts can be freely combined to leverage the design productivity. For example, Metropolis uses platform-based design (vertical concept), component-based design and application-architecture separation (horizontal concepts). They model heterogeneous systems at higher levels of abstraction by using the Metropolis metamodel specification where system functionality is presented by a set of objects that concurrently perform actions while communicating with each other. Jerraya et al. /2/, /4/, /17/ combine in /14/ the first two types of abstraction with the component-based design. An example of methodology implementation onto a SystemC simulation backbone with the support for OS scheduling is also given in /14/. SPACE methodology /3/ combines first two types of abstractions and component-based design.

C-extension languages, which deal with system-level modeling and HW/SW codesign, like SystemC /6/, /18/ or SpecC /19/, have support for TLM built in. Built around

the SpecC system-level design language /20/, Gajski et al. /7/, /13/ developed an SoC exploration methodology, which covers the entire path necessary for system development. Methodologies that base some of their work on these language extensions also favor variations of TLM, primarily computation-communication separation. The AAA /21/-/23/ methodology relies on the graph theory and focuses on automatic mapping of application to architecture. For manageability they use hierarchy, but to our knowledge, this methodology is weak in abstraction.

In order to enable early system-level exploration (where lower-level details are not captured yet) only the first and the third type of abstraction can be used. The second type does not support this because it requires all layers to be available if model evaluation is to be done. Taking into account popularity of TLM based abstraction approach (throughout the usage of SystemC or SpecC) we concentrated our research work on TLM based abstraction. Nevertheless, we believe the approach we propose can be successfully combined with other two types of abstraction.

3 Levels of abstraction within TLM exploration

The starting point in TLM is a functionally complete SW description of the algorithm that needs to be implemented in the system. This forms an un-timed model, called the specification model (presented with circled A in Fig. 1), usually described with a programming language (e.g. C). At this abstraction level, the model yields confirmation of functional correctness of the algorithm description. The functional correctness is later refined to include architectural timing details and component communication is specified in the way that is architecturally implementable. In the final step, a fully functional and fully timed RTL model of the system is obtained (circled F). A detailed explanation of TLM can be found in /6/ or /7/.

The purpose of following the steps of successive refinements is guiding the designer from the starting specification model (also called the system architecture model (SAM) /6/) through the well defined intermediate steps (i.e. levels of abstraction) to the final implementation model (referred to also as the RTL model). By making design decisions and detailing the system description, the designer is able to replace more abstract models with less abstract down to the point where the system description can be used for the system realization by the use of automation tools.

To study the concept of successive refinements supported within TLM, we focused on aspects this approach features. In Fig. 1 it can be observed that the TLM approach favors the description approach where the designer can independently focus on the *computation* and *communication* part of system modeling.

For the purpose of a clearer separation onto problem domains we examined the TLM from the perspective of the Rugby metamodel (RM) /8/, /11/. It has been developed especially for the study of modeling and handles abstraction in essentially more fundamental way. The RM introduces four domains; *computation*, *communication*, *data* and *time*. Terms *computation* and *communication* from the TLM model should not be regarded equal to terms having the same name in the RM. As it will be explained later, these TLM terms are a superset of terms in the RM.

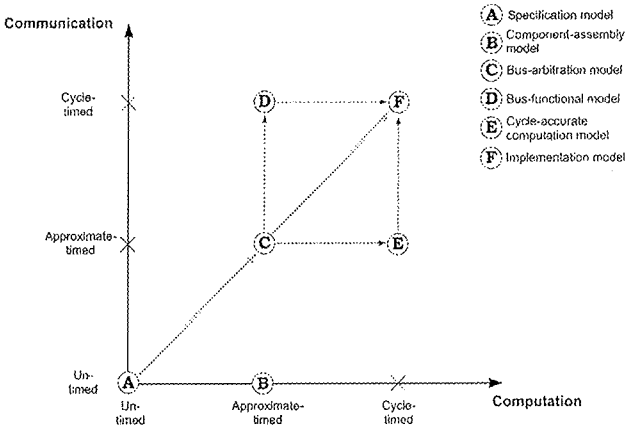

Fig. 1. The basic concept behind the TLM approach is successive refinement of system description

Table 1. Abstraction level exploration in the TLM model

TLM aspect	abstraction exploration of...			
	computation domain	*communication domain*	*data domain*	*time domain*
computation	partial	x	partial	full
communication	x	full	partial	full

3.1 TLM computation domain

We argue that methodologies offering a formal support from the TLM's specification model onward offer full utilization of abstraction in the time domain (RM) only. The TLM starting system description contains no information about computation durations, which implies a fully abstract time domain (RM). Throughout the TLM refinement steps, timings get gradually defined to a completely time-accurate model, thus lowering the level of abstraction.

On the contrary, we find the data (RM) and the computation (RM) abstraction of quite a low-abstraction level, as the starting model (TLM) is functionally complete and correct. The possible abstraction levels are therefore in these two domains not fully explored. Data to be processed and their processing algorithm (i.e. computation) are in TLM starting point already in their final form. Input test vectors for functional verification can be fully applied and all further refinements deal with timing and communication issues instead. If computation is to be remapped to the HW domain, this becomes the issue of transformation between

the same levels of abstraction, but across the heterogeneity boundary.

Computation row of Table 1 summarizes the above.

3.2 TLM communication domain

Here the communication (RM) and time (RM) domains are fully explored throughout the available levels of abstraction. If we take a closer look at the TLM's specification model communication principles, we can see that communication is simply implemented with a mechanism of a shared variable, additionally extended with events /6/, /7/. This kind of communication is obviously of a high abstraction level. Throughout the successive refinements, shared variables are gradually replaced with channels down to the point where every data and every handshake bit level signal of the wire on the communication bus are known. Simultaneously care is also taken for lowering the level of abstraction in the time domain (RM). When communication is abstracted with shared variables, communication takes no time, but throughout the successive refinements an accurate RTL implementation model is gradually obtained.

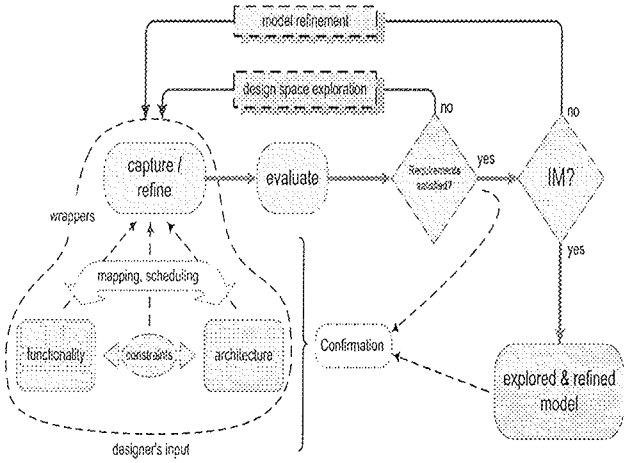

Fig. 2. The design flow

Minding the fact that the TLM system model communicates data produced in the computation part of the model, the abstraction level exploration of both data (RM) domains are equal (i.e. only partial).

The bottom row of Table 1 summarizes the above.

3.3 Downsides of the TLM approach

In the initial step of the TLM, a model has to be captured in the functionally complete manner in order to allow for the evaluation of high-level design decisions and profiling it for further refinements and design decisions. It is already an established practice to describe the model functionality in a programming language (e.g. C or C++). The major drawback of this approach is the starting point for the description of the algorithm which is, seen from the system perspective, insufficiently abstract. This puts a burden on system-level design alternatives exploration, because a low-level description needs to be included in the model of the system in order to obtain evaluation feedback. In the codesign process, binding ad-hoc decisions at early stages should be avoided as they unjustifiably narrow the available design space and eliminate potentially better design solutions.

Furthermore, the TLM SAM execution model is SW oriented and thus inherently sequential and memory based, while HW is fundamentally concurrent. Sequential algorithms that are proved as effective in SW are seldom the best choice in HW. As it has already been pointed out in literature (e.g. /25/), algorithms targeting HW mostly significantly differ from algorithms targeting SW due to the exploitation of different types of implementation resources. Making further design decisions based on profiling of the SW description (e.g. /13/, /26/) can not provide substantial information for optimal HW/SW codesign.

Even if the issue of inherently suboptimal solution is left aside, raising the level of abstraction addresses the large conceptual gap between the high-level system description and the programming language functional description. We propose a modeling solution at higher levels of abstraction that resembles formal methods of description of UML class diagrams /12/, which is further leveraged with a domain-specific semantics and syntax, promoting it to the form of a formal system specification. A formal high-level specification benefits from an executable specification throughout all layers of abstraction - not only functionally complete.

4 The design flow supporting higher levels of abstraction

The approach we propose enables a systematically and intuitively developed model of the system beginning with a description at higher levels of abstraction. This narrows the gap between the informal system specification and the initial formal model description. We provide techniques that enable creation of an executable and evaluatable model of high-level system description starting with pure abstract computation operations, operating on abstract data transmitted through abstract communication channels and taking no or evaluated amount of time. Sequences of transformations between abstraction levels feature smaller gaps, thereby extending a solid ground for future research in the field of automatic transition between successive stages. Important high-level design decisions are in this way fragmented into several smaller decisions, offering improved system level design guidance.

Following our methodology (Fig. 2), designing a system starts by capturing the system-level specifications about the system to be developed. The designer first identifies the system-level functionality, high-level proposition of an architecture and system restrictions and demands (constraints). Wrappers provide support for capturing function-

ality, constraints and architecture. From that an executable specification can be built at all levels of abstraction serving as a basis for analysis of the model. Even at a high abstraction level an executable model of the system is built. Its purpose is evaluation of system level aspects. Throughout lowering the level of abstraction the feedback information about design decisions is becoming more detailed.

After comparing evaluation feedback results with constraints, the designer gets confirmation if the system description is viable or not. If the evaluation results do not meet the specified constraints, changes need to be made in the system description until requirements are met. The process of finding a viable solution at the specific abstraction level is called design-space exploration, shown by an inner loop arrow in Fig. 2. Once the designer's input is found viable (i.e. system constraints are met), the design space is narrowed and the model description needs to be further detailed. Detailing is done by means of model refinement (outer loop arrow in Fig. 2) by means of abstraction principles given in previous text. The process of model refinement has to be done until the level of abstraction is low enough (i.e. the IM is reached), where automated tools can take over.

4.1 Wrappers

In order to enable transparent, concise and domain-separated capturing of the specification about a system to be developed, we provide a library of wrappers that promotes application of vertical and horizontal complexity-addressing concepts presented in Section 2. As a vertical complexity-addressing concept we provide techniques that allow capturing and evaluating the system-level information above the levels of abstraction fundamentally supported in TLM. Horizontal complexity-addressing concepts that our methodology supports are functionality-architecture and computation-communication separation. By applying domain separation, the designer can independently focus on each problem domain.

Besides domains separation and capturing and evaluation of the system information, the role of wrappers is also providing basis for fast and easy construction of executable and evaluatable model. Our methodology provides support for firmly based design decisions grounded on the model evaluation feedback results, as for example execution time, resource activity, idle time and utilization share. To relieve the designer of the burden of repeatedly implementing support for these commonly required aspects, we implemented support for logging relevant information as an integral part of wrappers. During model execution, wrapper's logging functionality is responsible for automatic collecting of information by means of simulation traces that can be later reviewed and from them important design decisions can be drawn.

System functionality is intuitively described as a network of tasks that communicate through ports, thus providing grounds for computation-communication separation. For the simulation and evaluation purposes we provide a modeling scheme where all instantiated functionality wrappers (representing tasks) are capable of operating in parallel. From the functionality point of view, the task execution is limited only by their data dependency. The maximum level of parallel operations that specific algorithm permits is first examined and this is later decreased by applying restrictions of execution units. Each task is assigned an execution unit responsible for characterizing the cost of executing services (e.g. time, energy, and size by means of logical block or transistor count) and providing limitations on executions (e.g. resource un/availability). When more than one task specifies the same execution unit, it is up to the execution unit scheduling policy to determine the outcome of such a request. This provides the basis for automatic mapping and scheduling algorithms in the way the design requirements are met.

4.2 Backbone for simulation and evaluation

To provide support for describing and evaluating an executable model of the system, we utilize mechanisms provided by the SystemC modeling language. We furthermore leverage them with the proposed concept of the heightened level of abstraction presented in previous sections. As shown in Fig. 3, we provide libraries of wrappers on top of SystemC library /6/, /18/. By instantiating and extending the wrappers with algorithm-specific features, the designer can rapidly capture and evaluate the system description at different levels of abstraction. The description of the system is compiled and executed and simulation traces are obtained. They are used as a basis for system analysis of the applied design decisions.

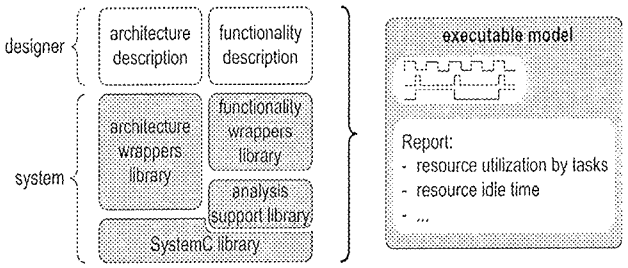

Fig. 3. Libraries utilization for evaluation

As depicted in Fig. 3, we built support for functionality wrappers tightly integrated with analysis support and underlying SystemC libraries. Actual implementation of computation is realized within functionality wrappers library. The role of architecture wrappers is to enable a rapid description of architectural resources responsible for services characterization and defining constraints on services required by the algorithm. Services offered by a given architecture are for example characterized by means of execution time, energy, area, etc. Complementary, constraints provide support for issues like for example resource contention and communication not being timeless, etc.

Support for analysis is automatically appended to every instantiated task and collects information relevant for evaluating the system description. After executing a model of the system, collected simulation traces and numerical data are available for analysis. The information collected is based on architecture services utilization, coupled with their constraints and characterization. Various system implementations can be easily built (exploring design space) and feedback results of their evaluations can be used for finding a path towards a solution that best meets system constraints. As it will be shown in the case study section, the information relevant for further design decisions (e.g. resource utilization or task being idle because of resource contention) is obtained automatically.

5 Case study – system-level JPEG model exploration

To put the concepts of the proposed heightened level of abstraction into practice, we present a system-level case study of a JPEG encoding system. A more detailed information about the JPEG standard can be found in /27/ and the complete C reference code in /28/. Presenting the entire design flow from the initial idea, over various high-level exploration aspects, down to the final implementation would be much out of the scope of this paper and would even blur the entire idea of the system-level modeling we propose. To our knowledge, the idea of the heightened level of abstraction as presented here has not been presented and studied elsewhere and so we are not able to make side by side comparison conclusions. However, the results we obtain can be proved correct if compared with full implementations, as for example /27/. Differentiation of our proposal essentially originates from the fact that concepts we propose help the designer at abstraction levels that are higher than the level of abstraction used in initial capturing model of the system of the state-of-the-art methodologies.

Fig. 4 depicts an implementation model of a JPEG encoding system targeting a single processor (*processor p1*) and one memory bus (*busUnit b*). The JPEG encoding process consists of four consecutive stages: color-space conversion (*rgb2yuv*), forward discrete cosine transform (*frwrdDCT*), coefficient quantization (*quantization*) and entropy coding (*entrCod*). Other pre and post processing stages (e.g. down-sampling) are omitted here for clarity. It is straightforward to model the JPEG encoding process in this way since it reflects an intuitive flow of data processing stages. According to the aforementioned processing stages we instantiated tasks with data dependency as indicated in Fig. 4, thus forming a formal description and a frame for further detailing. In contrast, Fig. 5 shows a slightly different implementation, offering more architectural resources, with three processors (*p1*, *p2* and *p3*) all connected to a shared memory bus (*busUnit b*).

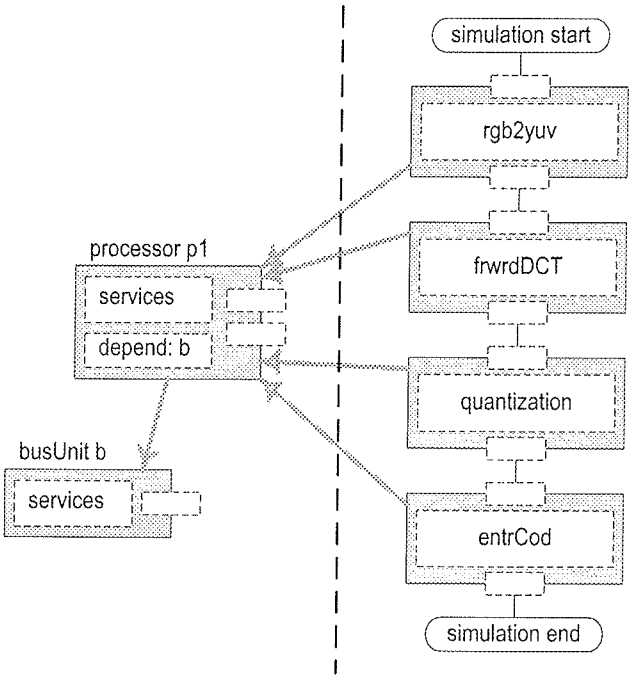

Fig. 4. JPEG encoding model of the system targeting single processor

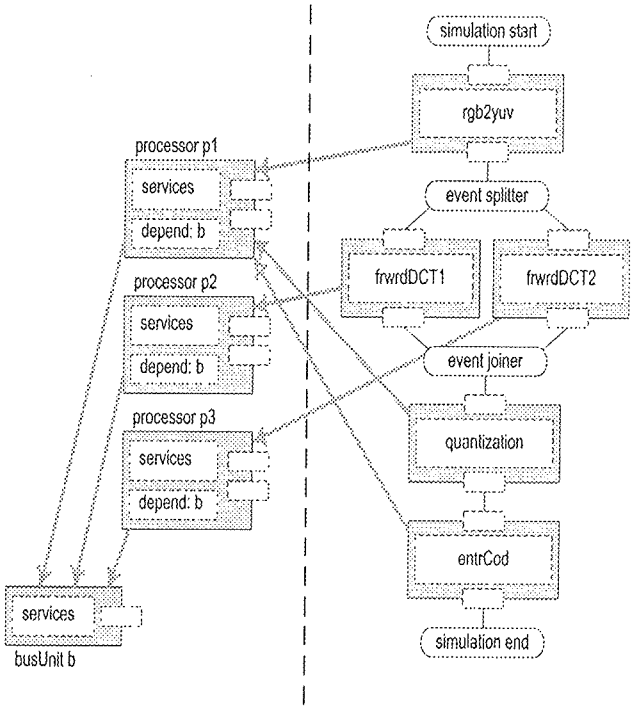

Fig. 5. JPEG encoding model of the system targeting three processors

To study the possibility of speedup at the system level, the designer needs to determine high-level properties of the algorithm; the minimum amount of data that can be processed between successive stages is an 8x8 pixel block and a complete color picture is composed of three color planes. To exploit these algorithm properties, we experimented with pipelining where individual color planes are

sequentially fed into the encoder, i.e. immediately after a stage processes one color plane, the data is forwarded to the next stage (firing job request). If based only on data dependency (i.e. without consideration of architectural limitations), the successive stage can start with processing immediately after the first chunk of data is processed in previous stages.

To allow distribution of the processing load in the case where more than one execution unit is available for execution, a simple event splitter is used. It is based on a round-robin scheduler, where jobs are alternatively assigned to *frwrdDCT1* (executed on *p2*) or *frwrdDCT2* (executed on *p3*). Similarly, the event joiner merges job requests from both tasks to execute *quantization* (executed on *p1*).

Fig. 6 lists a high-level algorithm description for the forward DCT encoding stage. When studying a specific DCT realization (e.g. /28/), it can be concluded that 11 multiplications and 20 additions are needed to apply the 1-D DCT transform. These operations represent abstract functions. The purpose of the triple nested *for* loops of Fig. 6 listing, explained from the inside out, is as follows. The 2-D DCT transform (of an 8x8 pixel block) is obtained by applying the 1-D DCT transform to rows first (loop 3) and columns second (loop 2). The pixels to be transformed must be grouped in segments of 8x8 pixels (*structured block* in data domain) (loop 1).

```
void frwrdDCT::MainThread()
{  // divide requested size (m_bytes) into blocks of 8x8 bytes and
   // do 2 pass   1D DCT - horiz.pass: 8B read, processing, 8B write
   // vert.pass: 64B read, processing, 64B write

   int blocks = ROUND_UP( m_bytes, 64 );

   if( m_bytes % 64 )
      cout  << sc_time_stamp()
            << " input size to frwrdDCT is not 8x8 aligned\n";

   for( int i = 0; i < blocks; i++ ) // loop 1
   {
      // pass 0 - horizontal, pass 1 - vertical
      for( int pass = 0; pass < 2; pass++ ) // loop 2
      {
         for( int line = 0; line < 8; line++ ) // loop 3
         {
            m_pExecUnit->GetData(this, (pass==0)?8:32, false);

            // 11MUL, 20ADD
            m_pExecUnit->Mult( 11 );
            m_pExecUnit->Add( 20 );

            m_pExecUnit->WriteData(this, (pass==0)?8:32, false);
} } } }
```

Fig. 6. High-level description of DCT. Although functionally incomplete, algorithm features are captured.

To further detail the forward DCT model, the accessing scheme for data storage can be applied. For this purpose, it must be defined how the pixels are stored in memory. Our choice was to store rows of pixels of one color plane into successive memory addresses. This implies that when reading rows (first pass), 8 subsequently read bytes are exactly what we need, but when reading columns (second pass), pixels are scattered in memory, and therefore more consumable memory accesses must be made. As the listing reveals, all requests are performed through execution unit interface (*m_pExecUnit*) attached to every functionality wrapper (functionality-architecture separation).

The models of the other JPEG encoding stages are constituted in a similar manner and won't be further detailed hereafter.

Fig. 7 shows a detailed explanation of a functionality wrapper whose responsibility is to provide common functionality for instantiating user-defined tasks, i.e. in this case study a user-defined encoding stages of Fig. 4 and Fig. 5. The functionality wrapper is provided within our high-level simulation library and it only needs to be extended with algorithm-specific features. It is responsible for accumulating input job requests (via *ExecuteThread event*), executing the user's algorithm (function call *MainThread()*) and signaling job completion (via *ThreadDone event*). Two execution modes are provided. In the first mode (*TriggerThread_in event* is unspecified) all accumulated job requests are executed sequentially without freeing the execution unit (architecture wrapper) between successive runs. In the second mode, a queued job is run only when *TriggerThread_in event* happens. This scheme offers a possibility for different scheduling types. In this case study we applied the first execution mode. The designer's sole responsibility is to implement the encoding stages via method implementation (*MainThread()*) that is executed through a functionality wrapper call.

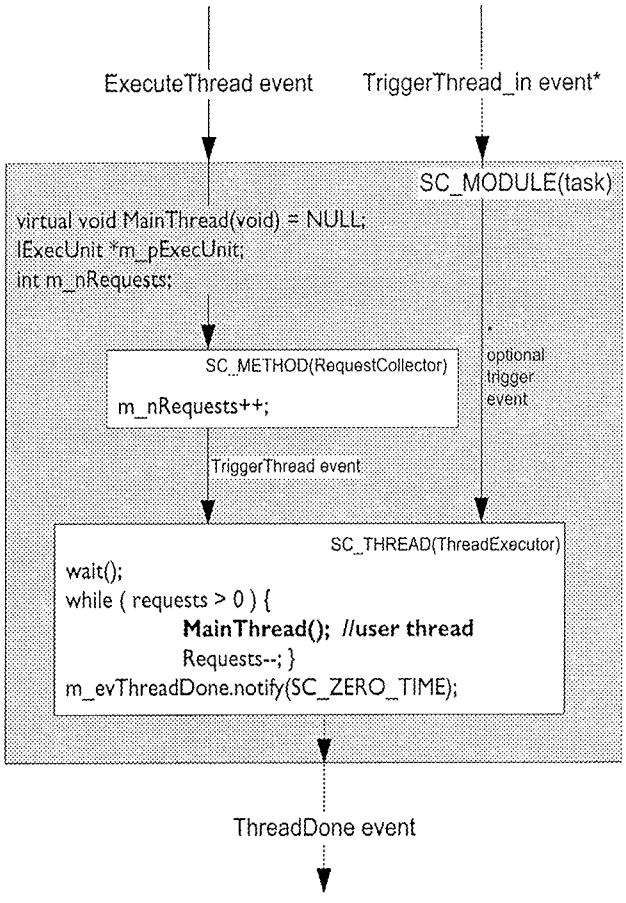

Fig. 7. Functionality wrapper

UML static class diagram notation of Fig. 8 illustrates how the model framework is applied to the JPEG encoding sys-

tem. As it can be seen, it is seamlessly connected with system-level modeling libraries we built. The bottom part of the diagram shows classes directly responsible for functionality implementation (*rgb2yuv*, *frwrdDCT*, *quantization*, *entrCod*) and architecture description (*processor*, *busUnit*). The system designer is responsible for the implementation of functionality methods. The top part of the diagram represents the system libraries built on top of SystemC. Decomposition into functionality and architecture can be clearly observed. Every task (implementing functionality as part of the algorithm) that needs to be described must be derived from the system functionality wrapper and must address all computational requests through the execution unit interface (*IExecUnit*). Computational requests are provided with the interface *IExecUnit* and must be implemented by the designer within the architectural description. For example, *processor* is derived from *IExecUnit* and the designer must specify all methods required by the interface. Following this rule, tasks can be simply mapped to their executors at the instantiation time and various mappings schemes can be explored. Following the same rules, the communication unit (*busUnit*) is created. Analysis support, built into the system libraries, provides automatic report generation. The class *unitStats* is responsible for collecting information about each architectural element (active or idle time, which task has occupied it, etc.).

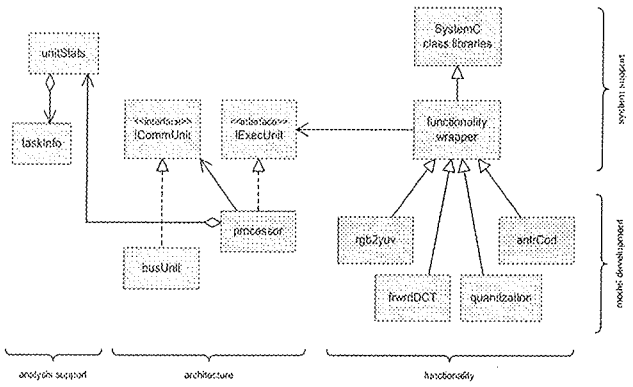

Fig. 8. Class diagram representation of the JPEG encoder model

5.1 Results

By following the rules and concepts established by our methodology, we built two different system specifications generating two executable models of the system. Results, generated automatically, exemplify the level of information obtained on the basis of a high-level system description. They represent a high-level design decision feedback that our methodology provides. Analysis of these results sets a solid basis for further design decisions. The most important results (automatically generated by *unitStats*) are presented in Table 2 and Table 3, presenting one- and three-processor variants. It should be noted that the task percentage of processor utilization in a single processor variant matches results stated in literature on the basis of a full functional implementation /27/. We obtained these results with the amount of around 100 lines of the source code for the system description (not taking into account our methodology system libraries built on top of the SystemC).

Table 2. Two processors

Architecture	p1		bus	
RET[ns]	430 680		93 240	
Functionality	exec. active	task wait	exec. active	task wait
rgb2yuv [% RET]	35.9	0	12.7	0
frwrdDCT [% RET]	46.8	23.9	61.8	0
quant [% RET]	13.5	31.2	18.5	0
entrCod [% RET]	3.8	9.0	7.0	0
total [ns]	430 680			

Each table is split into two parts: architectural part (upper) and functionality part (lower). The architectural part shows the resource execution time (RET), i.e. summation of the time the services were required from a specific resource. Complementary, the functionality part presents tasks' active and wait timings (in % of RET). The numbers stated in the *exec. active* column represent the percentage of the time a specific task was being actively executed on a specific resource. Similarly, the numbers stated in the *task wait* column represent the time a specific task had to wait a specific resource to become available – this is the time

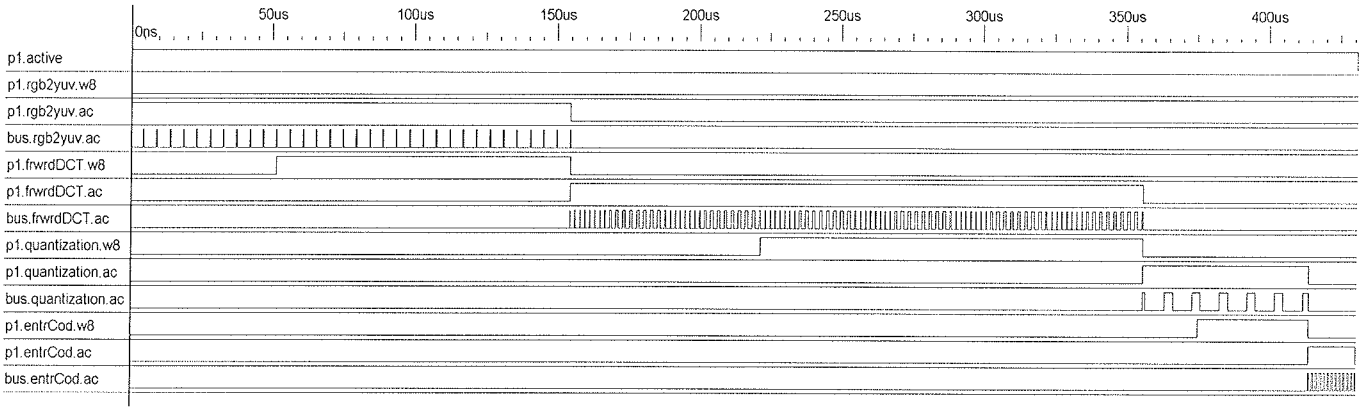

Fig. 9. JPEG encoding stage timing diagram (single processor)

Table 3. Three processors

Architecture	p1		p2		p3		bus	
RET[ns]	233 460		148 880		75 100		93 240	
Functionality	exec. active	task wait	exec. active	task wait	exec. active	task wait	exec. active	task wait
rgb2yuv [% RET]	66.7	0	-	-	-	-	12.7	1.4
frwrdDCT [% RET]	-	-	100	0	100	0	41.2 / 20.6[a]	15.5 / 8.5[a]
quant [% RET]	24.9	17.7	-	-	-	-	18.5	0
entrCod [% RET]	8.4	0	-	-	-	-	7.0	3.32
total [ns]	261 660							

[a] % RET originating from p2 and p3

interval during which a task may be executed regarding data dependency, but its execution is not started because of the unavailability of HW resources.

The second set of results is presented in Fig. 9 and Fig. 10, where the system activity is presented as a function of time. Fig. 9 clearly shows the execution of sequential stages and wait times (abbreviated as w8), consequently induced by pipelined data processing. Fig. 10 reveals that high-level descriptions with even very low complexity can quickly become difficult to analyze manually if parallel execution has to be considered. It can be seen that bus contentions arise, making the task execution time even less predictable. For example, this unpredictability is the reason why *p2* RET is not two times *p3* RET, despite that *p2* processes two color planes and *p3* just one.

In the above we presented design flow steps that enable capturing of high-level information about the system serving as a basis for building an executable and evaluatable model of the system. The model, once compiled and executed, provides an important feedback for further design decisions. After the satisfying high-level model of the system is obtained, the description of the system can be further detailed. Assured that important high-level decisions are formally verified, confidential migration towards an appropriate modeling level within TLM can be approached. The related research work can be used to take over from this point on.

6 Conclusion and future work

Our paper opens with an overview of contemporary research methodologies in order to present the reader the role of abstraction these methodologies apply. The use of abstraction is studied, because mastering transformations through abstraction layers is to our opinion the key to leveraging the design productivity. We propose an approach that primarily focuses on the levels of abstraction that are essentially higher than the widely used TLM. The concept of refinements for changing the level of abstraction is explained on the basis of the RM. By doing that, we come to the conclusion that there is still room for specifying levels of abstraction that are higher than in other current state-of-the-art modeling approaches.

In order to provide a framework for exploration at higher levels of abstraction we provide a set of libraries, built on top of the simulation kernel of SystemC, that enables rapid, transparent and formal capturing of high-level information about the system. Based on that, an executable code is obtained that simulates high-level aspects of the system under development. We also introduce an analysis support to provide automatic collection of the feedback data that serves as a basis for model evaluation and further design decisions. Our case study exemplifies the use of the methodology and shows how to quickly capture and transform the information at the proposed levels of abstraction into an executable and evaluatable model.

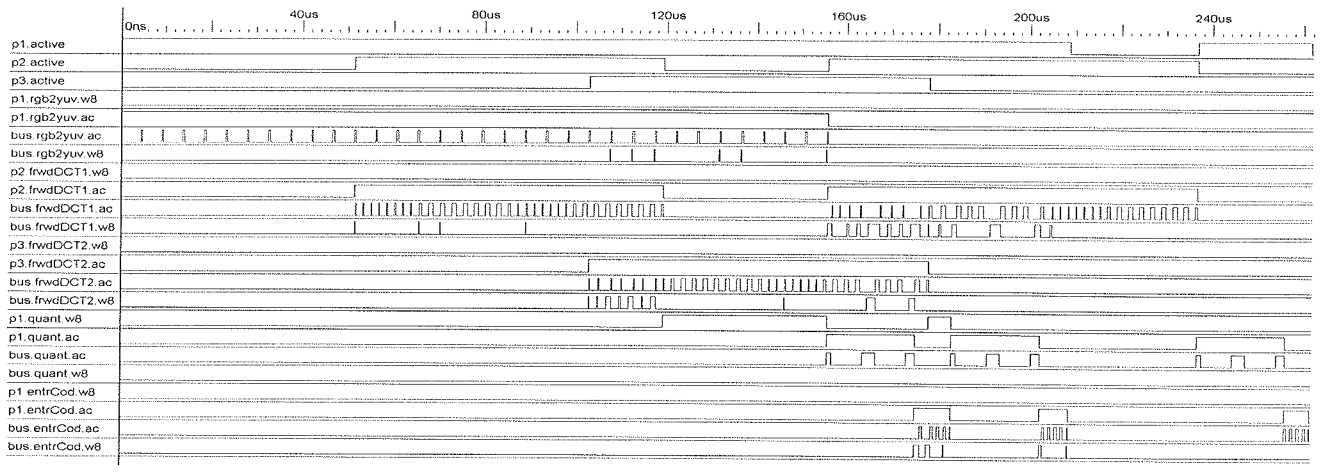

Fig. 10. JPEG encoding stage timing diagram (three processors)

Because of the regularities discovered in the process of building the high-level model of the system, in both the algorithm and architecture domain, we are currently engaged in graphical design capturing system together with the infrastructure for automatic code generation that we currently have to obtain by hand. Our long term plans involve implementing the design flow within the graphical modeling environment. Because of its rich extensibility, the GME modeling platform is considered as a modeling framework /29/, /30/.

7 Acknowledgment

The research was funded by the Ministry of Education, Science and Sport of the Republic of Slovenia through the program P2-0246-Algorithms and the optimization methods in telecommunications.

8 References

/1/ F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli. Metropolis: An Integrated Electronic System Design Environment. IEEE Computer Society, vol. 36, no. 4, pp. 45-52, April 2003.

/2/ W. O. Cesário et al. Multiprocessor SoC platforms: A component based design approach. IEEE Design and Test of Computers, Volume 19, Issue 6 (November 2002), pp: 52-63, ISSN: 0740-7475.

/3/ J. Chevalier, O. Benny, M. Rondonneau, G. Bois, M. Aboulhamid, F.-R. Boyer (2003). SPACE: A Hardware/Software SystemC modeling platform including an RTOS, Forum on Design Languages (FDL03), Frankfurt, Germany, 09/2003, pp. 704-715.

/4/ A. A. Jerraya, W. Wolf. Hardware/Software Interface Codesign for Embedded Systems. IEEE Computer Society, vol. 38, no. 2, pp. 63-69, February 2005.

/5/ L. Kaouane, M. Akil, Y. Sorel, T. Grandpierre. From algorithm graph specification to automatic synthesis of FPGA circuit: a seamless flow of graph transformations. FPL Proceedings, September 2003, pp. 934-943.

/6/ D. C. Black & J. Donovan. SystemC from the ground up. Springer, ISBN: 1402079885, June 2004

/7/ L. Cai, D. Gajski. Transaction Level Modeling: An Overview. CODES+ISSS'03, October 2003, Newport Beach, California, USA, Pp. 19-24, ISBN:1-58113-742-7.

/8/ A. Jantsch, S. Kumar, A. Hemani. Rugby: A Metamodel for Studying Concepts in Electronic System Design. Design & Test of Computers IEEE, July-September 2000 (Vol. 17, No. 3), pp. 78-85.

/9/ F. Balarin et al. Metropolis: A Design Environment for Heterogeneous Systems. In: A. Jerraya, W. Wolf, editors. "Multiprocessor Systems-on-Chips (The Morgan Kaufmann Series in Systems on Silicon)", Morgan Kaufmann, September 28, 2004, ISBN: 012385251X, pp. 357-393.

/10/ W. O. Cesario, A.A. Jerraya. Component-Based Design for Multiprocessor Systems-on-Chip. In: A. Jerraya, W. Wolf, editors. "Multiprocessor Systems-on-Chips (The Morgan Kaufmann Series in Systems on Silicon)", Morgan Kaufmann, September 28, 2004, ISBN: 012385251X, pp. 357-393.

/11/ A. Jantsch, S. Kumar, A. Hemani. The Rugby Model: A Conceptual Frame for the Study of Modelling, Analysis and Synthesis Concepts of Electronic Systems. p. 256, Design, Automation and Test in Europe (DATE '99), 1999.

/12/ OMG UML homepage: http://www.uml.org/

/13/ L. Cai, A. Gerstlauer, D. Gajski. Retargetable Profiling for Rapid, Early System Level Design Space Exploration. Proceedings of the 41st annual conference on Design automation, 2004, pp. 281 - 286, ISBN:1-58113-828-8.

/14/ A. Bouchhima, S. Yoo, A.A. Jerraya. Fast and Accurate Timed Execution of High Level Embedded Software Using HW/SW Interface Simulation Model. ASP-DAC 2004, Yokohama, Japan.

/15/ A. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis, M. Sgroi. Benefits and Challenges for Platform-Based Design. Proceedings of the 41st annual conference on Design automation (DAC '04), p.p. 409-414, 2004.

/16/ Metropolis homepage: http://www.gigascale.org/metropolis/

/17/ TIMA Laboratory, System-Level Synthesis Group: http://tima.imag.fr/sls/

/18/ SystemC OSCI homepage: http://www.systemc.org/

/19/ Center for Embedded Computer Systems (CECS) at UC Irvine: SpecC system: http://www.cecs.uci.edu/~specc/ SoC Environment - SCE: http://www.cecs.uci.edu/~cad/sce.html

/20/ SpecC Technology Open Consortium homepage: http://www.specc.org

/21/ AAA methodology and SynDEx homepage: http://www-rocq.inria.fr/syndex/

/22/ T. Grandpierre and Y. Sorel. From algorithm and architecture specifications to automatic generation of distributed real-time executives: A seamless flow of graphs transformations. First ACM&IEEE Intl. Conference on formal methods and models for codesign. MEMOCODE'03, Mont Saint-Michel, France, June 2003.

/23/ L. Kaouane, M. Akil, T. Grandpierre, Y. Sorel. A Methodology to Implement Real-Time Applications onto Reconfigurable Circuits. The Journal of Supercomputing, Volume 30, Number 3, December 2004, pp: 283 - 301.

/24/ A. A. Jerraya. Long Term Trends for Embedded System Design. EUROMICRO Symposium on Digital System Design (DSD 2004), Rennes, France, Sept. 2004.

/25/ B. Grattan, G. Stitt, and F. Vahid. Codesign-Extended Applications. 10th International Symposium on Hardware/Software CoDesign, pp. 1-6, Estes Park, CO, May 2002.

/26/ D. C. Suresh, W. A. Najjar J. Villareal, G. Stitt, F. Vahid. Profiling Tools for Hardware/Software Partitioning of Embedded Applications. Proc. ACM Symp. On Languages, Compilers and Tools for Embedded Systems (LCTES 2003), San Diego, CA, June 2003.

/27/ V. Bhaskaran, K. Konstantinides. Image and Video Compression Standards. Second Edition. Kluwer Academic Publishers, 1997.

/28/ Independent JPEG Group: http://www.ijg.org

/29/ GME project homepage: http://www.isis.vanderbilt.edu/Projects/gme

/30/ A. Ledeczi, et al. The Generic Modeling Environment. Workshop on Intelligent Signal Processing, Budapest, Hungary, May 17, 2001.

Jože Dedič, B.Sc.,
joze.dedic@fe.uni-lj.si,
Matjaž Finc, M.Sc.,
matjaz.finc@fe.uni-lj.si,
Assistant Prof., Dr. Andrej Trost,
andrej.trost@fe-uni-lj.si

University of Ljubljana
Faculty of Electrical Engineering,
Tržaška 25, 1000 Ljubljana, Slovenia
Tel: +386 1 4768 351

Prispelo (Arrived): 29. 05. 2006; Sprejeto (Accepted): 08. 09. 2006